2019

Metamorphic Testing with Input Patterns

Liqun Sun
University of Wollongong

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Metamorphic Testing with Input Patterns

Liqun Sun

This thesis is presented as part of the requirements for the conferral of the degree:

Master of Philosophy

Supervisor:
Dr. Zhi Quan (George) Zhou

Co-supervisor:
Dr. Casey Chow

The University of Wollongong
School of Computing and Information Technology

May, 2019

Declaration

I, *Liqun Sun*, declare that this thesis, submitted in partial fulfilment of the requirements for the conferral of the degree *Master of Philosophy*, from the University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualifications at any other academic institution.

Liqun Sun

May 28, 2019

Publications

1. L. Sun and Z. Q. Zhou, "Metamorphic Testing for Machine Translations: MT4MT," in Proceedings of the 25th Australasian Software Engineering Conference (ASWEC 2018), pp. 96-100. IEEE, 2018, doi: 10.1109/ASWEC.2018.00021.

2. Z. Q. Zhou and L. Sun, "Metamorphic Testing of Driverless Cars," Communications of the ACM, vol. 62, no. 3, pp. 61–67, March 2019. DOI: 10.1145/3241979

3. Z. Q. Zhou, L. Sun, T. Y. Chen, and D. Towey, "Metamorphic Relations for Enhancing System Understanding and Use," IEEE Transactions on Software Engineering, DOI: 10.1109/TSE.2018.2876433.

4. Z. Q. Zhou, L. Sun, T. Y. Chen, and D. Towey, "An Extended Abstract of: Metamorphic Relations for Enhancing System Understanding and Use," in Proceedings of the 41st International Conference on Software Engineering (ICSE '19), 2019.

Abstract

Despite endeavour to do their best to eliminate defects and to satisfy the end users as much as possible, developers hardly ever deliver faultless software. It is imperative for software developers to detect the defects and errors by software testing. However, for a long time, the oracle problem has often been an unavoidable challenge for software testing, which indicates that it is unlikely or unaffordable to verify the output given by a piece of software. To mitigate the oracle problem, metamorphic testing has become a popular method. Identification of effective metamorphic relations is critical when applying metamorphic testing. In most of the previous research, testers identify metamorphic relations manually and the way they do this is not systematic. Compared with identifying metamorphic relations case by case in most scenarios, deriving metamorphic relations from a high level of abstraction by a systematic methodology becomes an ambitious goal. For this purpose, this thesis propose a concept of metamorphic relation input patterns. A metamorphic relation input pattern can help both developers and end users to construct various concrete metamorphic relations to detect defects in software under test across many different application domains. Using this method, empirical studies have been further conducted using a variety of popular real-life services and applications. The present research proves the simplicity of the process and the wide applicability of the methodology by showing the detection of many previously unknown failures in the software under test.

Acknowledgments

It is incredibly difficult for an international student to do research in an unfamiliar country. Fortunately, with helping hands to overcome obstacles and difficulties, I finally completed my research.

Before starting a new chapter in my life, firstly, I would like to thank my supervisor Dr. Zhi Quan Zhou for the support in not only my study and research, but also in my daily life. I would not have been able to go the distance without his guidance, patience, intelligence and encouragement. I am also grateful to Dr. Casey Chow for kindly acting as my co-supervisor.

Secondly, I would like to thank my wife and my parents for supporting me spiritually throughout the course of my study in Australia.

Last, but not least, I would like to express my gratitude to my friends at the university for their many helps and tolerance.

Contents

Chapter 1

Introduction

1.1 Background

As software development and deployment grows more complex, in spite of strict implementations and skilled developers, delivering valid software of high quality is one of the most difficult goals. To a certain extent, software testing is seen as a 'last line of defence' against defects and is a widespread approach to determine whether the software satisfies expectations, specification and requirements. It is a widely-accepted practice for a business organisation to spend more than fifty percent of development costs on testing [1].

However, conducting software testing usually involves two intractable problems. Firstly, to evaluate quality and to find faults, testing generally uses a mechanism, namely oracle, against which testers can determine whether the outputs are correct for the software under test (SUT) [2]. The main difficulty in testing software is known as the oracle problem [2] which means the oracle is either expensive or impossible to be applied under certain circumstances. The research community considers this to be one of the fundamental challenges in software testing [3]. Secondly, running a program on a finite number of concrete test cases generally cannot prove the correctness of the program for the entire input domain, and it is practically not feasible to perform exhaustive testing. Metamorphic testing (MT) [4, 5] is a testing methodology that addresses the oracle problem and automated test case generation problem. A key activity of MT is the identification of metamorphic relations (MRs).

In this thesis, we propose a concept of metamorphic relation input patterns (MRIPs), with which multiple concrete MRs can be derived. We then use these MRs to assess a series of popular applications in the absence of an oracle.

1.2 Research Goals

This research has two main goals:

1. To study the pattern concept in the context of metamorphic testing in order to help stakeholders to derive concrete metamorphic relations that can determine whether software works properly across multiple scenarios and application domains.

2. To conduct empirical studies to evaluate the effectiveness of the proposed approaches.

1.3 Contributions of the Thesis

1. We develop a concept of a metamorphic relation input pattern, which can be used to derive multiple concrete metamorphic relations.

2. The empirical results show that our approach is easy to implement, effective to detect defects and is widely applicable to a range of application fields.

3. We extend metamorphic testing to help users to easily probe the characteristics of systems and make the systems serve their purpose with proper inputs.

1.4 Organisation of the Thesis

The remainder of this thesis is structured as follow. In Chapter 2, the literature on metamorphic testing and its application is reviewed. Chapter 3 proposes three metamorphic relation input patterns to derive multiple concrete metamorphic relations. In Chapters 4, 5 and 6, we conduct empirical studies by using derived MRs from three MRIPs to evaluate various applications. Chapter 7 presents the conclusion of this thesis and discusses the future directions for work in this area.

Chapter 2

Literature Review

2.1 Metamorphic Testing

2.1.1 Basic Concepts of Metamorphic Testing

A challenge in software testing is known as the oracle problem [2] which means the oracle is either expensive or impossible to be applied under certain circumstances. Metamorphic testing (MT) [6, 7, 8] has been proven to be able to effectively alleviate the oracle problem and test case generation problem. MT provides researchers and stakeholders a practical and powerful testing approach that is simple in concept and easy to implement, and has gained increasing popularity in recent years.

MT differs from conventional testing techniques in that it does not focus on the verification of each individual output of the software under test (SUT), but instead checks the *relations* among the inputs and outputs of *multiple* executions of the SUT. Such relations are called *metamorphic relations* (MRs). MRs are necessary properties of the intended program's functionality. If an MR is violated for certain test cases, then the SUT must be at fault. For example, consider a program $p(G,\ a,\ b)$ that identifies the shortest path from an origin node a to a destination node b in an undirected graph G. When G is large and complex, and when a and b are chosen randomly, it is hard to verify the output of p because of the lack of a tangible oracle. Nevertheless, some MRs can be identified for the shortest path problem. One of the MRs could state that swapping the origin and destination nodes should not affect the length of the shortest path [9]. Based on this MR, MT can be performed by running the program p twice: once for a *source execution*, on a *source input* $(G,\ a,\ b)$ to produce a *source output*; and once for a *follow-up execution*, on a *follow-up input* $(G,\ b,\ a)$ to produce a *follow-up output*. If a source output and its follow-up output are found to have different lengths, we say that the MR has been violated, and therefore a fault in p has been revealed. Many different MRs can be identified for the shortest path proble [9].

The increasing interest in MT is not only because of its ability to test software in the

absence of an oracle, but also because MT is based on a perspective not previously used by other testing techniques [10]. Researchers and practitioners have used MT to detect previously unknown software faults in various application domains [11, 12, 13, 14].

When studying MT for machine learning software, Xie et al. [15] found that an MR violation could reveal problems not only in the implementation but also in the choice of algorithm. In this situation, the MR was identified based on the expected functionality of the system: A violation of this MR, therefore, could indicate that the chosen algorithm was not appropriate (and not just that the implementation had faults). Xie et al.'s study was at the algorithm selection level, checking whether or not the adopted algorithm was appropriate. More recently, Zhou et al. [16] developed MT into a unified framework that covers verification, validation, and other types of software quality assessment. They showed that MRs could be identified by users based on their expectations (reflecting what they really cared about), rather than being based on the system designs chosen by the system designer or developer. Their study used MT at the top (system/services) level, and involved very large scale empirical studies with major web search engines.

The basic steps for MT as follows can be summarised [7]:

1. Identifying properties of the intended functionality of the SUT under test with necessary domain knowledge to construct metamorphic relations.

2. Build and run a test case generator to generate source test cases involved in the MRs; where an existing test suite is being used as the source test suite, the test case generator can simply take the existing test cases.

3. Build and run a test driver to execute the source test cases, generate follow-up test cases by referring to the prescribed MRs, and verify the test results against the MRs.

2.1.2 The Applications of Metamorphic Testing

Since the emergence of MT, much work has been done on various applications from different domains.

Segura et al. [7] highlighted the following areas where MT had been applied: Web services and applications, computer graphics, embedded systems, simulation and modelling, machine learning, bioinformatics, numerical programs (which only accounted for 5%), variability and decision support, compilers, components, autonomous vehicles, and others.

Jiang et al. [17] used metamorphic testing to support program repair without an oracle. In conventional automated programming, such as generate-and-validate techniques and correct-by-construction techniques, automated program repair (APR) based on test suites may have an oracle problem. Jiang et al.'s study mainly investigated the feasibility of combining metamorphic testing with APR and the repair effectiveness of this novel

approach. Jiang et al. used metamorphic testing as a checking mechanism to compare the outcome of individual outputs rather than using a test oracle. After conducting an empirical study on subject programs, IntroClass benchmark suite, which consists of 1143 C program versions, the results showed that this integration is both feasible and effective.

Sun et al. proposed an automated testing framework for Service-Oriented Application (SOA) to facilitate the testing of the orchestrated web services without an oracle from the perspective of service consumers [18]. The tester could use Web Services Description Language (WSDL) of the web service to construct metamorphic relations and configuration. The source test cases were generated by service description which recorded the previously executed tests or randomly from scratch. The follow-up test cases were generated by changing a source test case according to a metamorphic relation. Three web services with seeded faults were evaluated and the results showed that this approach was effective and efficient for implementing metamorphic testing.

Chan et al. [19] introduced a so-called metamorphic service on SOA. The SOA system in their study consistsed of a main service and other supportive services. Some metamorphic relations are linked between successful offline test cases and online supportive service computing. The results showed that when compared with the control experiment, the metamorphic service required less effort and detected more faults.

2.1.3 Metamorphic Testing from User Perspective

Based on the empirical experience of teaching software testing, it has been reported that metamorphic testing was suitable testing technique for end-users [20].

Liu et al. reported their experience of teaching metamorphic testing in a software testing subject [21]. The empirical study collected assignments from students at Swinburne University of Technology for three years. These students were from a variety of backgrounds, and had different career orientations. The results showed that the majority of students could understand the concept and identify metamorphic relations by themselves. Most students could develop automation testing with the lack of supporting tools and limited teaching time. However, test cases created by different entities may not show the same failure-detection effectiveness. Liu et al. suggested that it is important to find diverse metamorphic relations to improve failure-detection rates and also suggested a positive correlation between domain knowledge and good metamorphic relations.

Liu et al. conducted an in-depth study on the question “how effectively does metamorphic testing alleviate the oracle problem?” [22], and showed that, for the purpose of software verification, a small number of (on average, around six) diverse MRs were sufficient to match the fault-detection effectiveness of a test oracle for software implementing a knapsack algorithm.

2.1.4 Constructing and Selecting Metamorphic Relations

To a large extent, generation, selection and prioritisation of metamorphic relations dominates the effectiveness of metamorphic testing.

Chen et al. [9] were the first to investigate the nature of good MRs. They showed that if an MR's follow-up execution differs a lot from its source execution then this MR is likely to have a higher chance of revealing a failure. They did not quantitatively define the notion of "difference" or "dissimilarity" between source and follow-up executions, but pointed out that this could include, for example, execution paths, data flows, coverages, etc. In a follow-up study, Cao et al. [23] developed quantitative metrics for measuring the execution differences between source and follow-up executions, using the concepts of coverage Manhattan distance (CMD), frequency Manhattan distance (FMD), and frequency Hamming distance (FHD). The development of these metrics were inspired by Zhou et al.'s previous work on adaptive random test case prioritisation [24, 25]. Cao et al. investigated the correlation between the fault-detection effectiveness of metamorphic relations and the distance between the source and follow-up executions. Their results show that there is a significant positive correlation between the branch coverage Manhattan distance (BCMD) and the fault-detection capability of the MRs. They suggested a few ways for testers to apply their findings in practice.

Kanewala et al. reported an innovative approach to predict metamorphic relations using machine-learning techniques [26]. They created control flow graphs (CFG) from source code and extracted a set of features from the CFGs. Two algorithms, namely, support vector machine (SVM) and decision tree, were trained by these features. Their results showed that the metamorphic relations predicted by SVM were more effective for scientific computing applications. Additionally, their study suggested that small training sets could be doable for real-life testing. In recent work, Kanewala et al. [27] proposed a machine learning approach for automatic identification of metamorphic relations. Their approach uses graph kernels to evaluate the similarity among graphs. Their experiment involved six metamorphic relations and a corpus of 100 numerical functions. They assumed that functions are more likely to have similar metamorphic relations if they have similar operations. The empirical results show that the graph kernel using all paths in the CFG has the best prediction accuracy.

Chen et al. conducted a study on identifying metamorphic relations effectively and systematically using the category-choice framework [28]. In brief, they partitioned the input domain by category according to the specification, and this framework defines all possible combinations of inputs. A Java tool for this specification-based framework, namely MR-Gen, was designed to partly replace manual metamorphic relation identification. Three groups of participants were required to identify metamorphic relations for a parking fee system. The result showed that a tester using the tool could identify metamorphic relations

effectively and efficiently. The limitation of this study was the number and classification of participants. A total of 19 participants were involved and it was hard to distinguish the inexperienced from the experienced. In addition, it is not easy to get more real-life specifications from industry.

Gotlieb and Botella proposed a framework to automatically find test cases that violate an identified metamorphic relation [29]. This framework accepted programs that only consist of a single structured procedure and leveraged constraint logic programming techniques. The results showed that the prototype of this framework could work but inefficiently and that it would be impossible to conduct exhaustive search in a reasonable amount of time.

Ding et al. proposed a framework to iteratively generate metamorphic relations and test cases [30]. They refined the relations and tests by evaluating the test coverage, and mutation testing. A Monte Carlo simulation program was used to discuss the effectiveness of this approach.

2.1.5 Testing from User Perspective

Some empirical studies on web search engines were proposed to check both faults and user expectations. The metamorphic relations were conceived without deep knowledge of the underlying working mechanism of the search engines. A web search engine is the predominant gate which connects users and the Internet. Because of its complicated architecture and mass data, it is difficult to test search engines using conventional methods. Moreover, it is challenging to find an objective method to assess page ranking quality which involves user experience.

Zhou et al. presented an automatic metamorphic testing approach for web search engines, including Google, Yahoo! and Live Search [31, 32]. By querying with different operators or file types and comparing the source and follow-up outputs, search engine failures were detected. The results showed that the performance of these web search engines was not as good as users expected. This study also gave providers some suggestions about service quality.

More recently, Zhou et al. [16] developed a unified metamorphic testing framework for not only verification and validation but also other types of software quality assessment from user perspectives. They conducted large-scale empirical studies of major search engines including Google, Bing, Chinese version of Bing, and Baidu. In their study, the adopted the software quality model standard ISO/IEC 25010 for software quality assessment.

Segura et al. [33] further applied metamorphic testing to the domain of RESTful web APIs.

2.2 Summary

This chapter briefly reviewed some of the literature on different applications and aspects of metamorphic testing. Although the existing applications of metamorphic testing cover verification, validation and other types of assessment of software, most of the works focused on construction of MRs that are capable of detecting faults in a specific domain. The present thesis fills this gap by studying the identification and application of MRs without a domain limitation.

Chapter 3

Identification of Metamorphic Relation Input Patterns

3.1 Background

One of the most critical activities in all MT research is to identify effective MRs [6]. To apply MT, normally, testers are inclined to identify MRs from scratch case by case to initiate their work. This practice might be convenient in some situations but neither systematic nor efficient. To make this process more systematic, Zhou et al. were the first to propose an idea of using an abstract form of MR to derive multiple concrete MRs, and they called this abstract form of MR a "general metamorphic relation" [31, p. 3]. In the context of testing search engines, they described a "general metamorphic relation" as follows:

> MR_{SEARCH}: If search criterion X_2 implies search criterion X_1 (that is, if X_2 is satisfied then X_1 is satisfied), then $\#(X_2) \leq \#(X_1)$, where $\#(X)$ denotes the number of results satisfying the search criterion X.

Using this general MR, many concrete MRs can be derived [31], such as:

> MR_{OR}: If $A_1 \equiv (A_2$ OR $B)$, then $\#(A_2) \leq \#(A_1)$, where A_1, A_2, and B denote query conditions. A logical operator "OR" is used in this case for inclusive disjunction ($\vee$).
>
> MR_{AND}: If $A_1 \equiv (A_2$ AND $B)$, then $\#(A_1) \leq \#(A_2)$, where a logical operator "AND" is provided in this case for logical conjunction ($\wedge$).

In a follow-up work, a concept of "general relation" was reported by Zhou et al. to generalise the *subset* relation among the source and follow-up outputs [32]:

> $Pages(X_2) \subseteq Pages(X_1)$,

where $Pages(X_1)$ denotes the source output which meets the query criterion X_1 and $Pages(X_2)$ denotes the follow-up output which satisfies the search criterion X_2.

The results from empirical studies show the fault-detection effectiveness of the derived concrete MRs is at a high level [31, 32]. For example, when the user queried the Microsoft search engine with GLIF and then with GLIF OR 5Y4W, it respectively returned 11,783 and zero results. This apparently violated metamorphic relation MR_{OR} and, because the violation was reproducible, a defect in the search engine was identified.

Also using abstract MRs to derive concrete ones for systematic MR generation, Segura et al. [33] first introduced the term "pattern" into the MT research community. They explicitly proposed *metamorphic relation output pattern* (MROP) in the context of testing RESTful web APIs (that implement create, read, update, or delete operations over a resource). They stated that an MROP "defines an abstract output relation typically identified in Web APIs" and that "each MROP is defined in terms of set operations among test outputs such as equality, union, subset, or intersection." They found MROPs to be very helpful for deriving concrete MRs. More specifically, they identified six MROPs: (1) Equivalence (representing relations where the source and follow-up outputs include the same items although not necessarily in the same order); (2) Equality (representing relations where the source and follow-up outputs must contain the same items, in the same order); (3) Subset (representing subset relations among the source and follow-up outputs, similar to the "general relation" ($Pages(X_2) \subseteq Pages(X_1)$) proposed by Zhou et al. [32, p. 223]); (4) Disjoint (representing relations where the source and follow-up outputs should be disjoint sets—having no elements in common); (5) Complete (representing relations where the union of the follow-up outputs should contain the same items as the source output); and (6) Difference (representing relations where the source and follow-up outputs should differ in a specific set of items).

A concrete example of an "equivalence" MROP is that a search for YouTube videos should return the same set of videos "regardless of the ordering criteria specified (date, rating, relevance, title, or number of views)" [33]. Segura et al. [33] hypothesized that their proposed patterns could also be useful for automated inferencing of likely MRs for a given API; however, they also pointed out that such research, as in [34, 27, 35], was challenging, and still at an early stage.

Compared with Zhou et al.'s initial work on this topic [31, 32], Segura et al. [33] investigated the pattern concept explicitly and systematically, providing strong evidence of its potential usefulness and, hence, opening a new MT research direction into "metamorphic relation patterns" in a broad sense, as foreseen by Segura in his keynote at the third International Workshop on Metamorphic Testing (ICSE MET '18) [36].

Nevertheless, the work of both Zhou et al. [31, 32] and Segura et al. [33] faced a similar limitation: Their patterns were only designed for a specific type of program: Search engines [31, 32] or RESTful web APIs (performing one of four operations over

a resource—create, read, update, and delete) [33]. Furthermore, Segura et al. [33] only studied patterns for output relations, and the outputs must be sets. These limitations mean that all previous work [31, 32, 33] has a significant applicability problem.

In this thesis, we propose a concept of metamorphic relation input pattern (MRIP) that has no such restrictions of application domains.

3.2 Changing Direction

We first define a metamorphic relation input pattern (MRIP) as follows:

> *Definition 1*: A *metamorphic relation input pattern* (MRIP) is an abstraction that characterises the relations among the source and follow-up inputs of a set of (possibly infinitely many) metamorphic relations.

We next define the first MRIP, ”change direction.”

> *Definition 2*: The *change direction* MRIP refers to the existence of a *direction* element in the source input, either physical or logical, explicit or implicit, which can be changed to construct the follow-up input.

The “change direction” MRIP can be used to probe a system from different viewpoints, which will be useful for users and testers. The present research investigates 65 top commercial websites by inspecting very common website features, namely *search* and *sort*. Although the *sort* function may involve many filters and recommendation algorithms, this study only focuses on the “direction” element regarding the user input, or more specifically, the sort criterion (from high price to low price, or from low price to high price). We also test the “direction” element in Google Maps navigation service. From a user perspective, the direction is from the start point to the destination. Similarly, the “direction” element in a Google Maps location-based search is the direction from one source location to a target location. We also tested face recognition functions of three popular applications. These applications allow users to upload photographs that may include human faces. The “direction” element of the input is the direction of the x-axis of 2D images. Finally, we tested the Google Cloud Video Intelligence. It is a video analysis service that makes videos searchable and discoverable. The “direction” element indicates whether a video is played forwards or in reverse.

3.3 Replacement

The second MRIP we propose is named ”replacement,” defined as follows:

> *Definition 3*: The replacement MRIP refers to the construction of the follow-up input by replacing part of the source input.

The "replacement" MRIP can provide fundamental insights into the association between input and output in some systems. If we replace the value of a relatively independent entity in a system's input, only a corresponding part in the system's output is expected to be changed. The logic behind it is that some changes to the input should not impact the unrelated parts or the structure of the output. It is reported that a risk assessment tool called COMPAS, which is used to predict the risk of recidivism, is allegedly biased against black defendants [37]. The algorithm behind the tool is expected to make fair decisions regardless of skin colour. In other words, ethnicity is replaceable in this case. For software testing, an example of using the replacement MRIP is given as follows: The colours of vehicles should not have an impact on the recognition of these vehicles for a self-driving car. Therefore, testers can replace an object's colour and expect the same classification result. A fault in the obstacle recognition module is revealed if it classifies a red object as a car in the first test and then classifies the same object as a tree in the second test when the colour of the object is changed to green.

3.4 Perturbation

We define our third MRIP, perturbation, as follows:

> *Definition 4*: The perturbation MRIP refers to the construction of the follow-up input by applying small changes (perturbations) to the source input.

Almeida et al. studied Web Services for governmental applications and proposed six perturbation operations to mutate SOAP messages [38]. The empirical results show that test cases derived from these transformations were effective in real-life web services. However, their technique needs a tester to determine whether the outputs are correct or not. In the present study, we investigate the use of the perturbation MRIP for a different application domain in the absence of an oracle. Our result verification is fully automated.

Chapter 4

Empirical Evaluation of the Changing Direction MRIP

Five metamorphic relations derived from MRIP of Changing Direction are proposed to apply to a variety of application domains, including commercial websites, web mapping service, video analysis, open source software and social media.

4.1 Case Study of 65 Top Commercial Websites

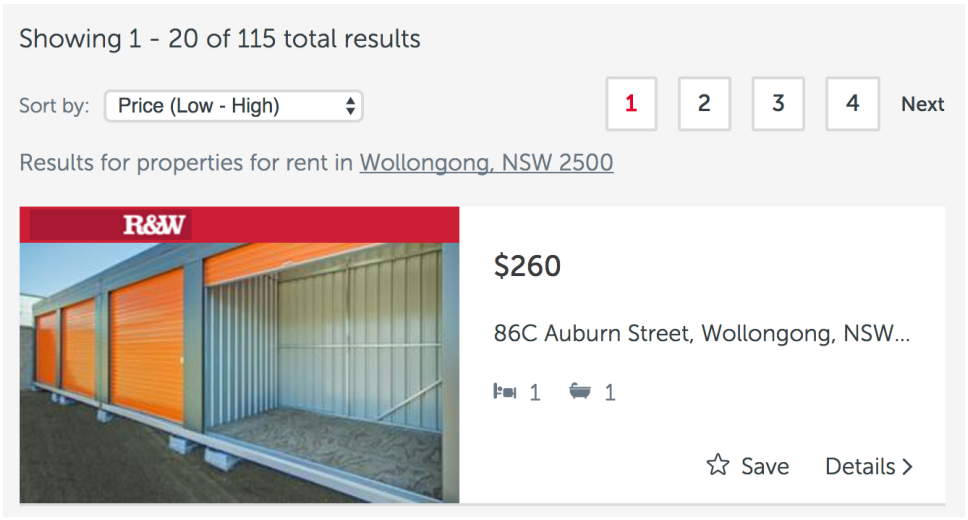
A website is sophisticated software which runs a range of programming languages. User experience largely depends on its functional correctness and usability. Because of the lack of reliable oracle (because of complex architecture and dynamic content), it is generally difficult to verify and validate a modern website with conventional testing methods. Metamorphic testing is an alternative testing approach to alleviate the oracle problem in assessing websites. This section takes advantage of MRIP to evaluate 65 popular commercial websites from the user perspective. The results show that most of the websites failed to provide reliable price sorting to the users. Corresponding countermeasures are also presented to mitigate the defects and to improve user experience.

4.1.1 Problems in Commercial Websites: A Motivating Example

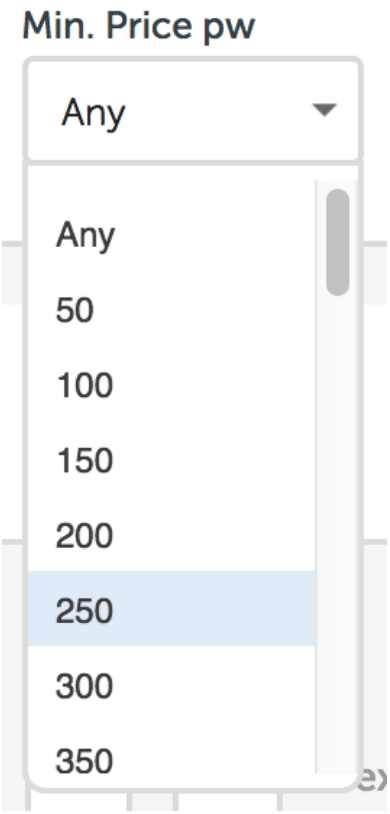
As an example of using MT on a commercial website, consider www.realestate.com.au and search rental properties. More than 5.5 million Australians visited realestate.com.au during February 2016 [39]. Currently, it is Australia's No.1 advertising website for real estate according to Alexa traffic ranking [40].

Firstly, we choose "Rent", uncheck "Surrounding suburbs", keep other filters with default, use "Wollongong, NSW 2500" as query terms and the system returns 115 results for properties for rent in Wollongong. After sorting the list by "Price (Low - High)", the first property "$260 86C Auburn Street, Wollongong, NSW" should be the cheapest of

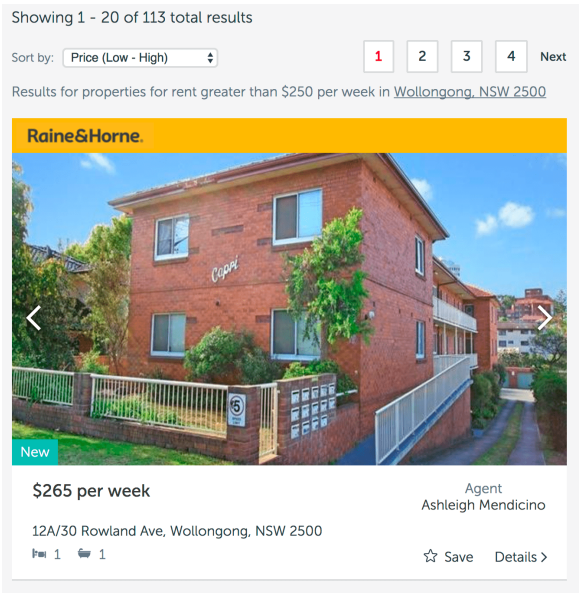
the results (see Figure 4.1(a)). Then we set the filter "Min. Price pw" with 250(see Figure 4.1(b)), there are fewer results and the first property changes to "$260 12A/30 Rowland Ave, Wollongong, NSW" (see Figure 4.1(c)). This reveals a fault in the system.



(a) Results sorted by price



(b) Price filter

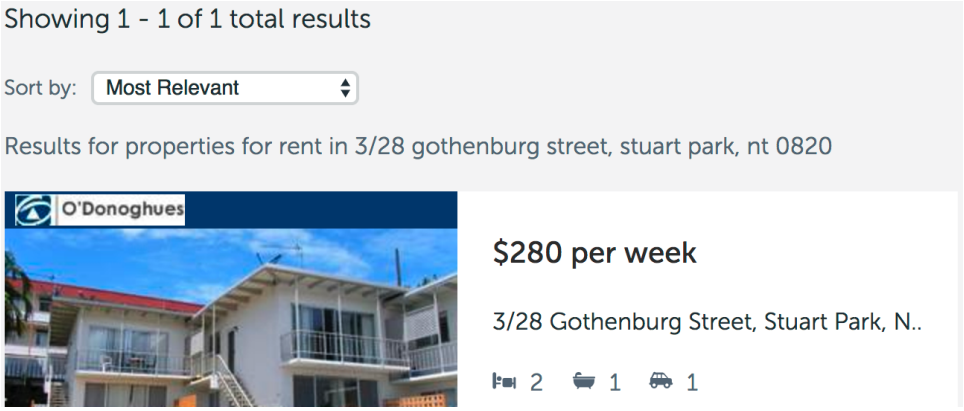


(c) Results filtered and sorted by price

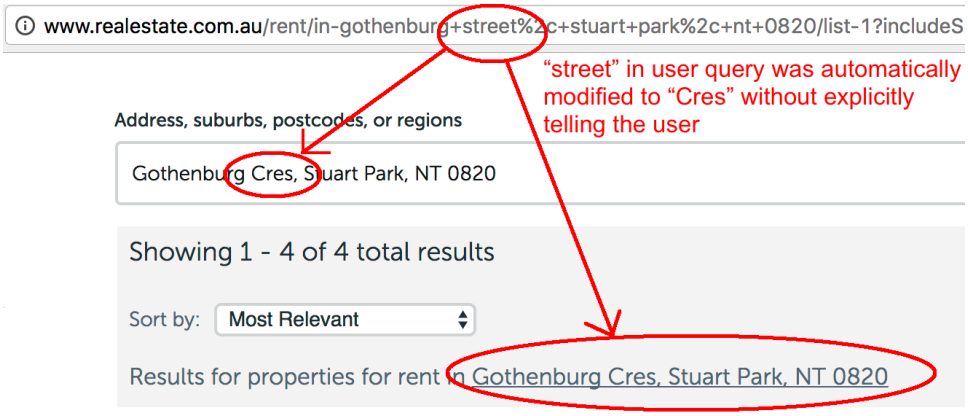
Figure 4.1: Fault in price sorting on realestate.com.au

Secondly, we choose "Rent", uncheck "Surrounding suburb", keep other filters with default, use "3/28 Gothenburg Street, Stuart Park, NT 0820" as query terms, the system gives the match feedback for this address (see Figure 4.2(a)). Then we find all rental

properties on this street by using "Gothenburg Street, Stuart Park, NT 0820", we expect the results include the previous one. Nevertheless, it shows the results for properties for rent in "Gothenburg Cres, Stuart Park, NT 0820" (see Figure 4.2(b)). We find the system somehow automatically replace the query terms. It is noteworthy that the original query terms could be seen in the URL.



(a) Excerpt of the results page for query "3/28 Gothenburg Street, Stuart Park, NT 0820"



(b) Excerpt of the results page for query "Gothenburg Street, Stuart Park, NT 0820": All four results are wrong as they do not match user query.

Figure 4.2: Fault in searching on realestate.com.au

The above examples show that a popular commercial website can have multiple bugs, easily causing erroneous search results that may bring losses to users when they are unaware of these errors in the retrieved information.

4.1.2 Objectives of the Experiment

Sixty-five commercial websites were chosen from a range of major ranking lists on the basis for either popularity or annual revenue. There are 35 retailing websites from "Global Powers of Retailing 2017" [41] which is an annual report that identifies the top 50 largest e-retailers around the world regarding annual revenue in 2015. There are eleven real estate websites from "eBizMBA top 15 Realestate" [42] and eight car sales websites from

"eBizMBA top 15 Car" [43]. eBizMBA is a guide which identifies the top 15 most popular websites in their category rank that are derived from average of each website's Alexa Global Traffic Rank and the U.S. Traffic Rank from Quantcast. There are 11 pharmacy websites from "Alexa Top Sites by Pharmacy" [44] which is a web traffic rank for online pharmacies provided by Alexa. Only websites which published in English whose features include price sorting were used in this study.

4.1.3 Derived Metamorphic Relations

$MR_{PriceSort}$: Let $s(q,c)$ denote a $perfect$ query function, where q denotes a query criterion and c denotes a sorting criterion. We removed all other query conditions and the results are supposed to be completely sorted according to c. If $c1$ and $c2$ denote the criteria that sorts the results by price in ascending and descending order respectively, then $s(q,c1)$ and $s(q,c2)$ should return exactly the same set of results, but in reverse order.

4.1.4 Experimental Design

An experiment was conducted on each website with $MR_{PriceSort}$ which includes one source test case and one follow-up test case. It is noted that the sequence of execution does not impact the results. The source and follow-up test cases used different direction, sorting by price in ascending and descending order but populated exactly the same query term as the text input. It would be perfect for comparison if all 65 websites use the same query term but this was impossible because of the different businesses associated with them. For example, real estate websites would collect user's location information about the areas interested in to generate reasonable results but a fashion websites do not ask for location information. To minimise the number of query terms, the 65 websites were grouped into nine categories (see Table 4.1) according to their business types. To make sure query terms were suitable for all websites within the same category, the query terms were selected manually. All websites within the same category were performed using the corresponding query term (see Table 4.1). For instance, the query term "source" was used to test grocery websites and "oil" was used to test pharmacy websites. Another standard is that they need to return search results with at least one record. However, we found that some websites did not respond with results for a general keyword. This strategy is probably driven by commercial considerations. For example, www.sears.com only showed an advertisement page when the search term used was "women." Therefore, we used "women dress" and the website returned a list containing expected goods. For those websites that could not return results, a backup query term was generated for them. They are listed in the third column of Table 4.1.

Category	$Query_1$:1	$Query_1$:2
body care retailer	body	
car sales	BMW(zip:11223)	
consumer electronics retailer	mouse	
department retailer	women	women dress
groceries	sauce	
home improvement	knife	
office supply	pencil	
online pharmacy	oil	
online realestate	brooklyn,NY	New York,NY

Table 4.1: Query terms

4.1.5 Experimental Results

Table 4.2 summarizes the results of all 65 websites under test. Columns #3, #4, #5, #6 and #7 of Table 4.2 show five types of MR violation, namely count consistency, completeness of result, separate section, same price reverse and different price reverse. Only five websites (www.hsn.com, www.asos.com, www.tesco.com, mexmeds4you.com, www.pharmacy2u.co.uk) did not violate MR.

In this section, the observations, causes and their relationship with software quality model standard ISO/IEC 25010 [45] are discussed.

4.1.6 Type 1 of MR Violation: Count Inconsistency

In a normal running system, the result counts should be the same If there are no changes of the items in the results from the source test case. A lack of count consistency implies at least one of the price sorting result count is inaccurate. The precision of the result count is related to the *functional correctness* (under *functional suitability*), which refers to the "degree to which a product or system provides the correct results with the needed degree of precision".

Observations

The result counts between the output of source and the follow-up test case were compared. A total of five websites were count inconsistent. For instance, searching "women dress" in amazon.com returned 822726 results (see Figure 4.3(a)) for price low to high and 822742 results (see Figure 4.3(b)) for price high to low. Not only is there inconsistency in total count, but also counts in filters is sometimes different. When we searched "women" in target.com, four source test case and follow-up test case counts in size group filter were

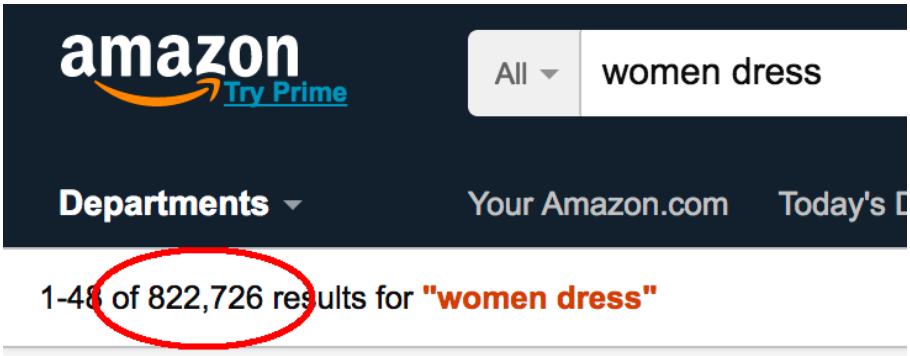
	1	2	3	4	5	6	7
	Website	**Category**	**Type 1 count consistency**	**Type 2 completeness of results**	**Type 3 separate section**	**Type 4 same price reverse**	**Type 5 different price reverse**
1	www.bathandbodyworks.com	body care retailer	✓	✓	-	x	✓
	www.autotrader.com	car sales	✓	✓	bottom: "no price available";	x	✓
2					featured/premium/standard		
	www.carfax.com	car sales	✓	x	bottom: "call for price"	x	✓
					"Other cars with free CARFAX		
3					Reports"		
4	www.cargurus.com	car sales	✓	x	"featured list", "standard list"	x	✓
5	www.carmax.com	car sales	✓	✓	-	x	✓
6	www.cars.com	car sales	✓	x	bottom: "not priced"	x	✓
7	www.carsdirect.com	car sales	✓	x	bottom: "contact seller"	x	x
8	www.kbb.com	car sales	✓	x	top: "private seller" bottom:"price N/A"	x	✓
	www.thecarconnection.com	car sales	✓	✓	bottom: "no pricing",	x	✓
9					"featured listing", "standard list"		
10	www.bestbuy.com	consumer electronics retailer	✓	✓	-	x	✓
11	www.currys.co.uk	consumer electronics retailer	✓	✓	-	x	✓
12	www.amazon.com	department retailer	x	x	bottom:current unavaliable	x	x
13	www.costco.com	department retailer	✓	✓	-	x	✓
14	**www.hsn.com**	**department retailer**	**✓**	**✓**	**-**	**✓**	**✓**
15	www.jcpenney.com	department retailer	✓	✓	-	x	✓
16	www.johnlewis.com	department retailer	✓	✓	-	x	✓
17	www.joybuy.com	department retailer	✓	✓	-	x	x
18	www.kohls.com	department retailer	✓	✓	-	x	x
19	www.llbean.com	department retailer	✓	✓	-	x	x
20	www.macys.com	department retailer	✓	✓	-	x	x
21	www.overstock.com	department retailer	✓	x	-	x	x
22	www.qvc.com	department retailer	✓	✓	-	x	✓
23	www.sears.com	department retailer	✓	x	-	x	✓
24	www.target.com	department retailer	x	x	-	x	✓
25	www.very.co.uk	department retailer	✓	✓	-	x	x
26	www.walmart.com	department retailer	x	x	-	x	✓
27	www.wayfair.com	department retailer	✓	✓	-	x	x
28	bedfordfair.blair.com	fashion retailer	✓	✓	-	x	x
29	shop.nordstrom.com	fashion retailer	✓	✓	-	x	x
30	store.nike.com	fashion retailer	✓	✓	-	x	x
31	**www.asos.com**	**fashion retailer**	**✓**	**✓**	**-**	**✓**	**✓**
32	www.hm.com	fashion retailer	✓	✓	-	x	✓
33	www.neimanmarcus.com	fashion retailer	✓	✓	-	x	✓
34	www.next.co.uk	fashion retailer	✓	✓	-	x	✓
35	www.zalando.co.uk	fashion retailer	✓	✓	-	x	✓

Table 4.2: Results of Experiments (to be continued)

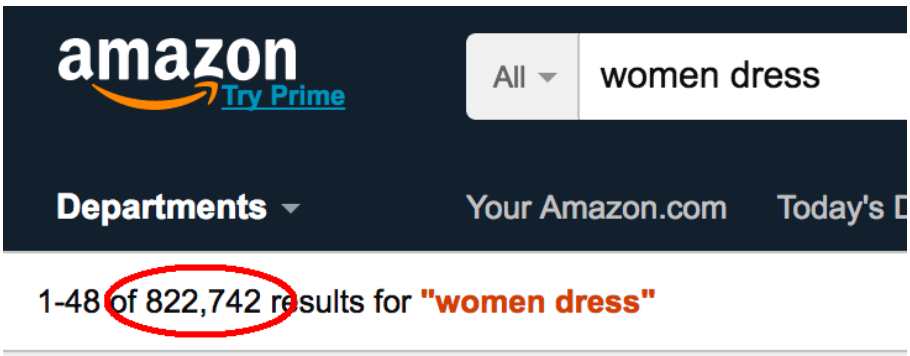
	1	2	3	4	5	6	7
	Website	**Category**	**Type 1 count consistency**	**Type 2 completeness of results**	**Type 3 separate section**	**Type 4 same price reverse**	**Type 5 different price reverse**
36	www.ocado.com	groceries	✓	✓	bottom: out of stock	x	✓
37	www.sainsburys.co.uk	groceries	✓	✓	-	x	✓
38	**www.tesco.com**	**groceries**	**✓**	**✓**	**-**	**✓**	**✓**
39	www.habitat.co.uk	home improvement	✓	✓	-	x	✓
40	www.homedepot.com	home improvement	✓	x	-	x	x
41	www.williams-sonoma.com	home improvement	✓	✓	bottom:unrating items	x	✓
42	www.staples.com	office supply	✓	x	bottom:out of stock online	x	✓
43	www.viking-direct.co.uk	office supply	✓	✓	bottom: "current unavaliable"	x	✓
44	medichest.com	online pharmacy	✓	✓	-	x	x
45	**mexmeds4you.com**	**online pharmacy**	**✓**	**✓**	**-**	**✓**	**✓**
46	well.ca	online pharmacy	✓	✓	bottom: "sold out"	x	✓
47	www.cincottachemist.com.au	online pharmacy	✓	✓	-	x	x
48	www.cvs.com	online pharmacy	✓	✓	-	x	✓
49	www.epharmacy.com.au	online pharmacy	✓	✓	-	x	✓
50	www.medshopexpress.com	online pharmacy	✓	✓	-	x	x
51	www.netpharmacy.co.nz	online pharmacy	✓	✓	-	x	✓
52	**www.pharmacy2u.co.uk**	**online pharmacy**	**✓**	**✓**	**-**	**N/A**	**✓**
53	www.pharmcom.com	online pharmacy	✓	✓	-	x	✓
54	www.walgreens.com	online pharmacy	✓	✓	bottom: "priced per store"	x	✓
55	hotpads.com	online realestate	✓	x	-	x	x
56	www.apartmentguide.com	online realestate	✓	✓	-	x	x
57	www.apartments.com	online realestate	x	x	bottom:"no availability"	x	✓
58	www.homes.com	online realestate	✓	✓	-	x	✓
59	www.realtor.com	online realestate	✓	✓	-	x	✓
60	www.redfin.com	online realestate	✓	x	-	✓	✓
61	www.remax.com	online realestate	✓	✓	-	x	✓
62	www.rent.com	online realestate	✓	✓	bottom: "contact for pricing"	x	x
63	www.trulia.com	online realestate	✓	x	-	x	✓
64	www.zillow.com	online realestate	x	x	-	x	✓
65	www.ziprealty.com	online realestate	✓	✓	-	x	✓
	Inconsistency rate (%)		**7.69**	**26.15**	**24.62**	**92.19**	**29.23**

Table 4.3: Results of Experiments (continued)

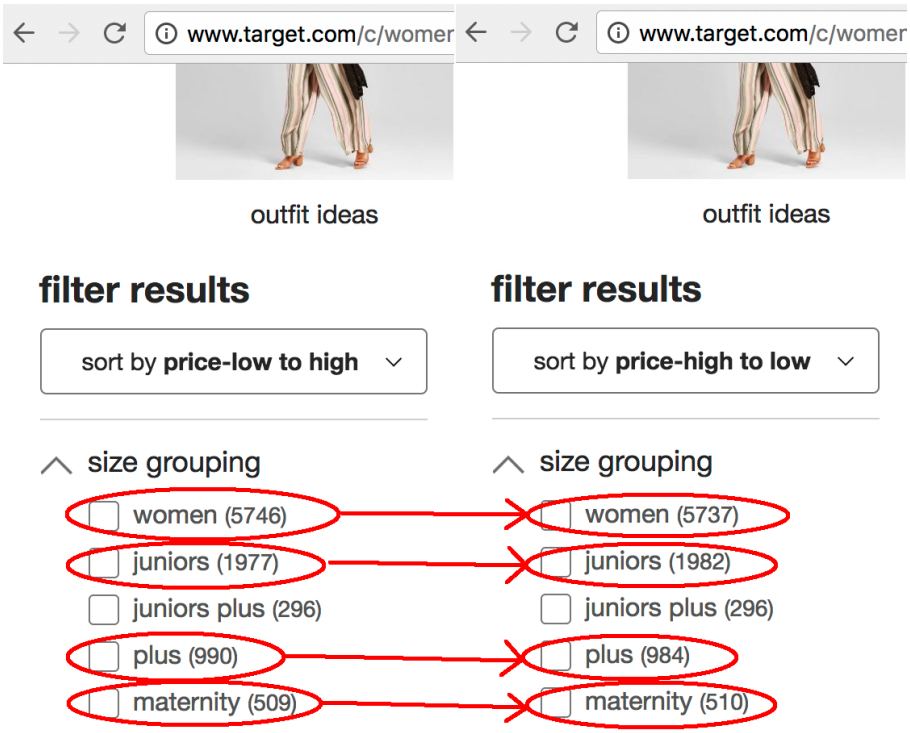
different (see Figure 4.3(c)). In the experiment, the count consistency was repeatedly verified to ensure that the inaccuracy of the result count occurred without any changes of the related items.



(a) Search "women dress" on amazon.com, sorting by price low to high



(b) Search "women dress" on amazon.com, sorting by price high to low



(c) Search "women" on target.com

Figure 4.3: Type 1 of MR Violation: Count inconsistency on amazon.com & target.com

Cause Analysis

Firstly, consider the consistency between the result count and the quantity of retrieved results. The result count on the web page is usually used to show the exact number of results. Only two websites, namely, sears.com and hotpads.com, displayed the estimated result counts "500+" and "28.9k", respectively. There are three causes for the inconsistency between them: the algorithm that obtains the count of results has a defect; the result count represents an outdated count of results, or the result count is the statistic of a different data source. The second of the causes may emerge in a system with a cache component because the calculation of the result count could be expensive for a large scale of data and this design could reduce the load on the system resource by tolerating the inconsistency to some degree. The third cause is related to a system with scaling issues. For example, the data source is from a master database and the count is calculated based on a slave database. During the experiments, we ensured that all detected failures were repeatable, and this practice effectively excluded the possibility of false positives caused by database update in the server side.

The count inconsistency between output of source and the follow-up test case may be explained in three ways: the algorithm may behave differently for sorting ascending or descending, the generated time of count in the cache may not be synchronised for sorting ascending or descending, or if the system uses different data source for sorting ascending or descending, the asynchrony between two data source leads to count inconsistency.

4.1.7 Type 2 of MR Violation: Incompleteness of Search Results

Incompleteness of search results is revealed by comparing the result count with the number of items in the result set. Part of the result set has no chance of being shown because of truncation. Furthermore, in a price sorting scenario, this fault may mislead the users who want to determine the lowest price at the end of the result set by sorting price in descending order. This constraint is related to the *capacity* (under *performance efficiency*), which refers to the "degree to which the maximum limits of a product or system parameter meet requirements" [45].

Observations

After comparing the total count of results with the results list, 17 websites that did not satisfy the completeness of search results were found. These websites have a limitation on the number of returned results and truncates the results list when the total number of the results exceeds the limitation. The sizes limit varied from hundreds to tens of thousands. For example, overstock.com only returned a maximum 1020 results but it showed 421637 items found for "women" (see Figure 4.4).

Cause Analysis

In this study, all websites with this fault have a pre-defined maximum size for a result set and this limitation may vary from hundreds to tens of thousands. Table scan for search and sorting operation which needs numerous intermediate data, involve many I/O activities [46]. A constraint, the number of items in a result set is generally used to improve the performance.

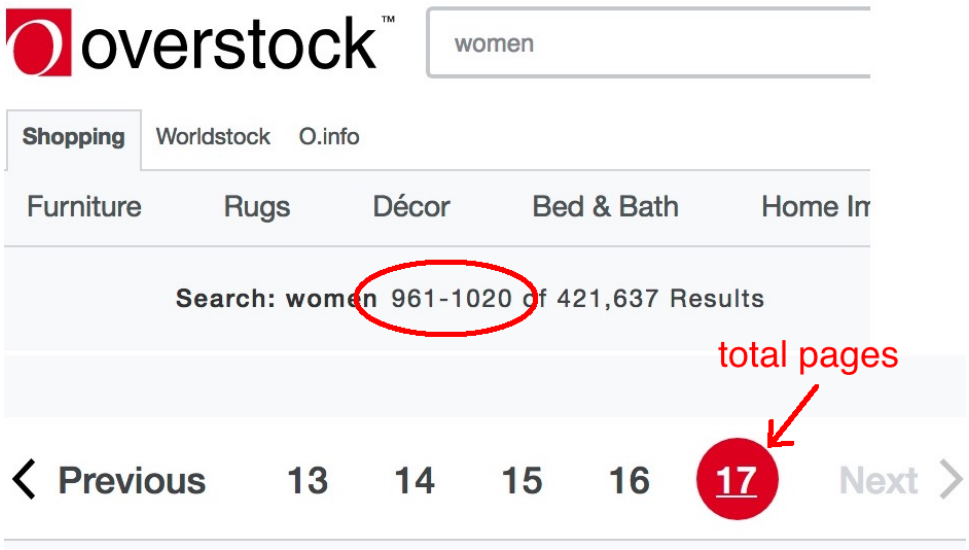


Figure 4.4: Type 2 of MR Violation: Incompleteness of results on overstock.com

4.1.8 Type 3 of MR Violation: Separate Section

We find some items always stay at the beginning or end of the result set. They belong to separate sections which are independent of price sorting. Consequently, users cannot directly find the lowest and highest price from first item in ascending and descending order, respectively. This requirement for extra effort is related to *functional appropriateness*, which refers to the "degree to which the functions facilitate the accomplishment of specified tasks and objectives" [45] .

Observations

A total of 16 websites were found to not sort all results together and to have a fixed section (top or bottom) to separate the results. For example, rent.com always keeps "contact for pricing" items at the bottom of list regardless of descending or ascending order (see Figure 4.5).

Cause Analysis

Some websites prefer to place featured items at the beginning of the result set and leave low priority items, such as "out of stock" and "price unavailable", at the end regardless of price sorting. The top position serves as an advertisement position and the items at the end of list may be unavailable to users but still need to be searched. The effectiveness of this business strategy is not part of this present study. However, from a user's perspective, this strategy makes the price sorting fail to fulfil the purpose of its operation.

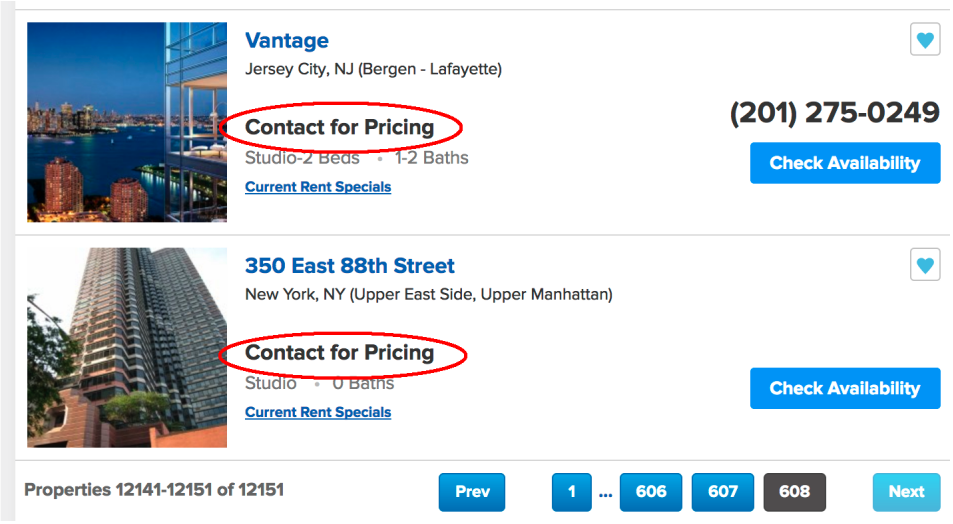


Figure 4.5: Type 3 of MR Violation: Separate section on rent.com

4.1.9 Type 4 of MR Violation: Same Price Not Reverse

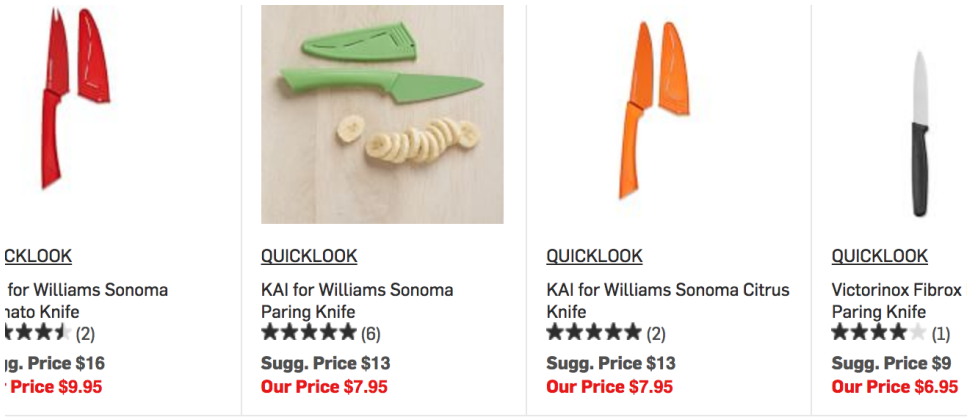
Most websites used in this study do not reverse order for same price items even though price sorting only involves the order of different prices. Nevertheless, it has a negative impact on both the users and the website. Let us suppose that there are four items (A,B,C,D) with price $10, $9, $9 and $8. The list shows "A", "B", "C", "D" when it is sorted by price in descending order. It shows "D", "B", "C", "A" when it is sorted by price in ascending order. High exposure of a certain product is unfair to other products. Additionally, users review the list and stop at "B" and then reverse the list and may stop at "B" because they think the entire list has been reviewed.

Observations

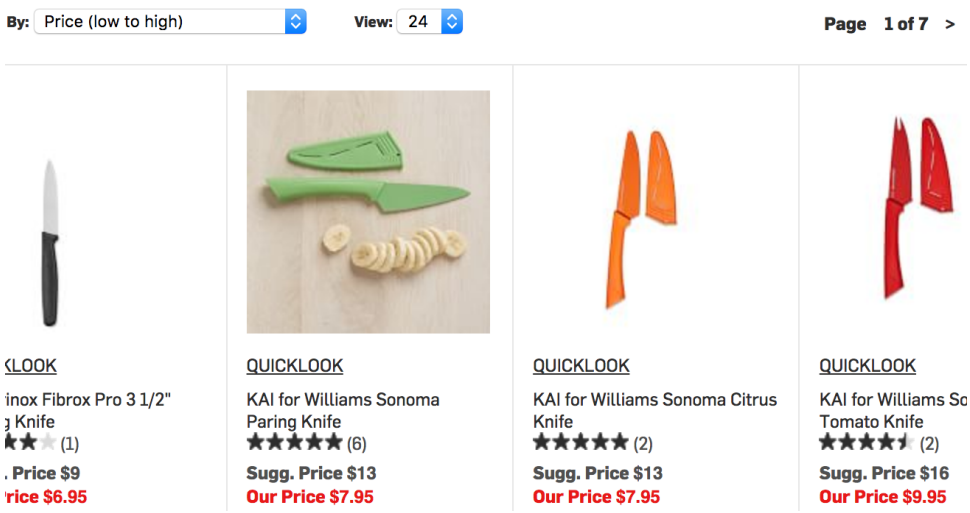
Only five websites reverse order for items with the same price. Most commercial websites keep the same sequence for the same priced items after reversing price order. For example, a $7.95 knife on williams-sonoma.com keeps the same order (see Figure 4.6). This could not be determined for one website (www.pharmacy2u.co.uk) because no items of the same price are shown in the list.

Cause Analysis

Websites that have this fault may sort a result set on multiple properties and price is the first priority when users perform price sorting. From a user perspective, the property of the second priority is hidden and cannot be changed. As a result, the "price sorting" is not just purely sorting by price.



(a) Search "knife" on williams-sonoma.com, sorting by price high to low



(b) Search "knife" on williams-sonoma.com, sorting by price low to high

Figure 4.6: Type 4 of MR Violation: Fail to reverse same price items on williams-sonoma.com

4.1.10 Type 5 of MR Violation: Different Price Not Reverse

Sorting price is a process of arranging items in a numerical descending or ascending order. The violation of sequence for different price is the most serious fault of all categories. Users cannot trust the lowest and highest prices, but they also cannot refer to any part of the sequence. This fault is also related to the *functional correctness*.

Observations

Because of the truncation of large result lists by some of the websites as explained previously, it was decided to use more specific queries (as shown in Table 4.4) to investigate "different price reverse". Nineteen websites do not sort different price properly. For example, first page sorting by price low to high shows $7.99 is the lowest price (see Figure 4.7(a)), but last page sorting by price high to low shows $11.53 is the lowest price (see Figure 4.7(b)). It reveals a system default in sorting function.

Category	$Query_2$:1	$Query_2$:2	$Query_2$:3
body care retailer	soap		
car sales	BMW 328i (zip:11223)	BMW 3 series Gran Turismo (zip:11223)	BMW 3-series (zip:11223)
consumer electronics retailer	laptop		
department retailer	women dress	percolator	
groceries	oil		
home improvement	cement tile		
office supply	dell all in one printer		
online pharmacy	zinc		
online realestate	zip:11223		

Table 4.4: Query terms for truncated result list

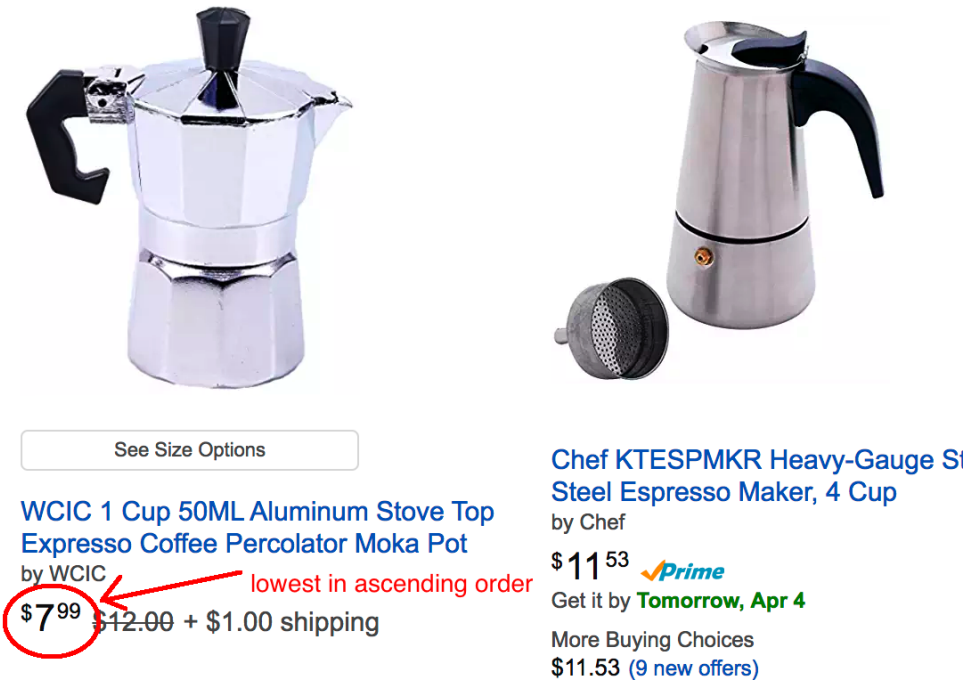
Cause Analysis

There are three causes for this fault: some items are groups consisting of several prices and the website chooses different prices for the item while sorting different orders, incorrect implement of sorting, for example, only sorting the first page of list or code does not works, different sorting based on different data sources that are not synchronised.

4.1.11 Recommended User Countermeasures

If the result count is from an outdated cache, the user could obtain the count by calculating the result list. In this research, the smallest number of count inconsistency is over 8000. This suggests that users could narrow down the search scope to avoid count inconsistency.

To avoid being frustrated by type 2 and 3 MR violation, users could obtain a completed result set by merging the output of source and follow-up test case. On the other hand, users should sort price in descending and ascending order to find highest and lowest price items at the beginning of the result set, respectively



(a) Search "percolator" on amazon.com, sorting by price low to high, first page

(b) Search "percolator" on amazon.com, sorting by price high to low, last page

Figure 4.7: Type 1 of MR Violation: Fail to reverse different price items on amazon.com

If same price items always keep a fixed sequence, it is better for users to go through all the same price items to overcome type 4 MR violations. In a scenario of finding the lowest and highest price in the result set, if a system cannot sort a list by price properly, users could repeatedly leverage the price filter, which function is assumed normal, by setting minimum and maximum prices according to the first and last items in the result

set, respectively, until desirable results are achieved.

4.2 Case Study of Google Maps Navigation

4.2.1 Objectives of the Experiment

In this section, the study shows that direction is not only the essential nature of a sorting algorithm, but also a navigation software's core.

Navigation software is designed to plan an optimal path from a starting point to a destination. Users can include extra conditions, such as travel mode and time. The generated path provides a series of instructions for the user to go to the target location. On one hand, Navigation software is unsurprisingly attractive to Internet user and it is installed on more than 50% of smart phones in the world [14]. On the other hand, the failure of navigation systems may be fatal because any error in this kind of software may result in significant accidents, especially with autonomous cars, and delivery drones becoming more and more popular around the world.

Although navigation software is both widespread and critical, the oracle problem still disturbs them just as it does other software. Because of infinite possible paths, it is difficult to verify whether the instructions created by the software are the most optimal option for user. In this section, this study demonstrate how the approach can assist non-professionals to estimate the returned results on maps.google.com (Google Maps web version). The popularity of Google Maps is unrivalled among all mapping services (except in China, where many Google services are not accessible) [14].

4.2.2 Derived Metamorphic Relations

$MR_{ReverseDirection}$: When using navigation software, a "direction" element can be defined as the direction from the starting point to the target location. Hence, a concrete MR can be derived from the "change direction" MRIP: *swapping the starting point and target location should generate a path with a very similar cost*. It should be pointed out that the cost of a path is about distance or time consumed (the equivalence in MR means *either* similar distances *or* similar times). The "time" is an estimated time without taking road conditions into account.

4.2.3 Experimental Design

In this study, the travel method selected was walking. This is because compared to other travel methods, such as driving and taking a bus, walking can effectively avoid one-way roads which is very common in the city. An investigation discovered very few one-way

roads for pedestrians. Because of unstable networks and web pages, all test results were manually validated.

In our experiments, Hong Kong and London, as the world's most visited cities, were selected as the candidate cities. One thousand pairs of valid addresses were generated for each city (some are in English and others are in Chinese). Request were sent to Google Maps for each pair of addresses (A, B), to get the path from A to B and then from B to A for walking. The outputs were then compared and under normal circumstances, the distances and time durations of the returned paths were expected to be similar. We used Google Maps default settings in all experiments. Furthermore, we did not sign-in any related accounts to because any personalised settings in the browser may influence the results.

4.2.4 Experimental Results

For London, 42% of the test cases returned different durations and 8% returned different distances. The corresponding rates for Hong Kong were 72% and 16%. According to the statistics, most differences were ignorable. However, some remarkable cases were discovered that might reveal a defect in the software. Two examples are introduced in Figure 4.8.

Figure 4.8a illustrates that, given a starting point (Portcullis House) and a destination (67 Bridge St), user will be given a path with a distance of "3 ft" [a]; It was a surprise to see that swapping the source and destination returned a path of "0.5" miles (Figure 4.8b) [b]. The algorithm may have its own legitimate reason to return these two paths. From the users' perspective, the difference between the two paths is unacceptable. Moreover, as shown in Figure 4.8b, it is said that "This route has restricted usage or includes private roads." In this case, the user is provided a route that may make the travel difficult.

Figure 4.8c [c] and Figure 4.8d [d] show a test case in Hong Kong which is very similar to the example in London.

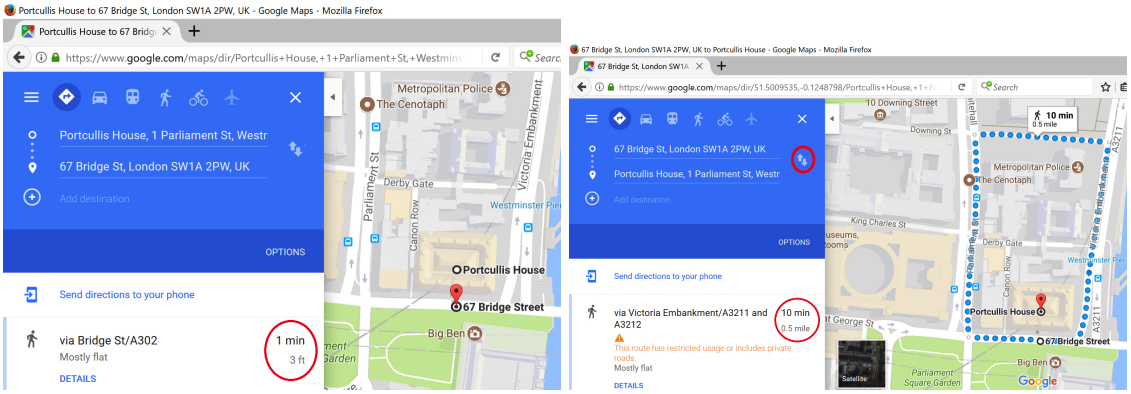
---

[a] URL: https://www.google.com/maps/dir/Portcullis+House,+1+Parliament+St,+Westminster,+London+SW1A+2JR,+UK/51.5009535,-0.1248798/@51.5013344,-0.1252238,17z/data=!4m8!4m7!1m5!1m1!1s0x487604c449a7df5d:0xe7a541ace7056d37!2m2!1d-0.1248698!2d51.5012679!1m0

[b] URL: https://www.google.com/maps/dir/51.5009535,-0.1248798/Portcullis+House,+1+Parliament+St,+Westminster,+London+SW1A+2JR,+UK/@51.5019238,-0.1271069,17z/data=!3m1!4b1!4m8!4m7!1m0!1m5!1m1!1s0x487604c449a7df5d:0xe7a541ace7056d37!2m2!1d-0.1248698!2d51.5012679

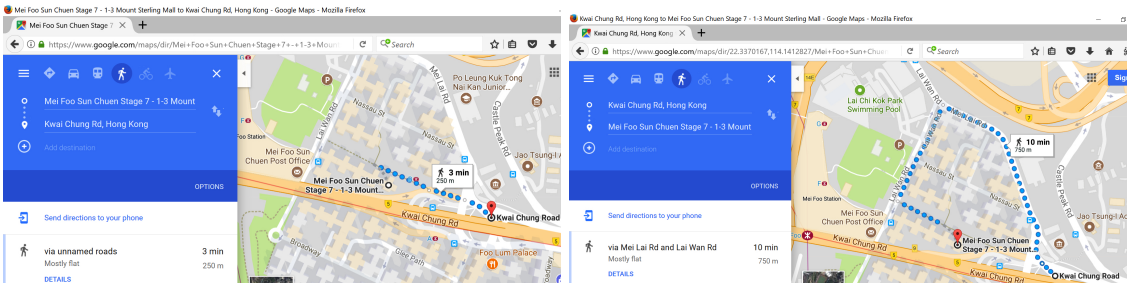
[c] URL: https://www.google.com/maps/dir/Mei+Foo+Sun+Chuen+Stage+7+-+1-3+Mount+Sterling+Mall,+1-3+Mount+Sterling+Mall,+Hong+Kong/22.3370167,114.1412827/@22.3374341,114.1379989,17z/data=!3m1!4b1!4m9!4m8!1m5!1m1!1s0x3403f8ab6af1cdfb:0x5d48e5358a826c08!2m21d114.1393603!2d22.3375777!1m0!3e2

[d] URL: https://www.google.com/maps/dir/22.3370167,114.1412827/Mei+Foo+Sun+Chuen+Stage+7+-+1-3+Mount+Sterling+Mall,+1-3+Mount+Sterling+Mall,+Hong+Kong/@22.3385225,114.1375424,17z/data=!3m1!4b1!4m9!4m8!1m0!1m5!1m1!1s0x3403f8ab6af1cdfb:0x5d48e5358a826c08!2m2!1d114.1393603!2d22.3375777!3e2



(a) Walking navigation in London: 3 ft, 1 min.

(b) Bad case detected: MR violation after reversing origin and destination: 0.5 miles, 10 min.



(c) Walking navigation in Hong Kong: 250 m, 3 min.

(d) Bad case detected: MR violation after reversing origin and destination: 750 m, 10 min.

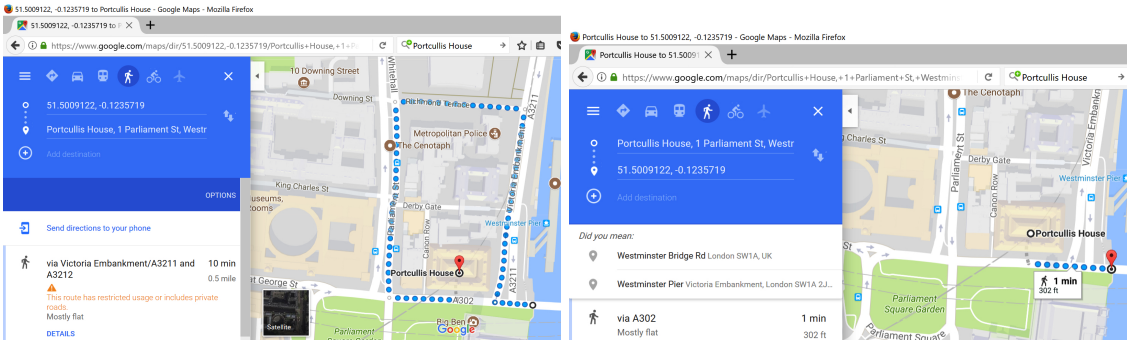
Figure 4.8: A "change direction" MR revealed Google Maps navigation defects in London (a&b) and Hong Kong (c&d)

4.2.5 Further Analysis

In all experiments, we used the default settings in Google Maps. To improve accuracy and make the experiments repeatable, real-time traffic data were disabled. To avoid personalised results, we did not log into any online accounts, including those for Google. All of the returned paths were checked to ensure that they did not have any one-way limitations for walking. It was determined that, following a "change direction" MR, users can easily find a better path using the icon shown on the page (such as those shown in Figure 4.8b and 4.8d). Without further information about the software, the reason why the software returned the very large difference between paths cannot be determined and the short one is obviously suitable to walk to both directions (Figure 4.8b and 4.8d). In other words, as a countermeasure, the user is recommended to select the shortest paths given by Figure 4.8a and 4.8c regardless of the direction.

This study has more meaning than just helping a user better use navigation software. The methods suggested could be simply used to optimise any existing navigation software without changing its design and architecture. Someone may argue that such an obvious bad case as that shown in Figure 4.8b is meaningless because the user can verify and

refuse the results immediately. However, some scenarios, such as self-driving cars and delivery robots have no human to check the rationality of path. Even if the system could do an exhaustive check and find the best path, the resource is unaffordable.



(a) The same bad case as in Figure 4.8b but with a slightly different starting point.

(b) The bad case was detected and the shortcut was found by swapping the origin and destination, using the same "change direction" MR.

Figure 4.9: Detecting the same bad case in another path, in London.

4.3 Case Study of Location-based Search

A different type of search product was investigated and the queries included both text and geographic information. Generally speaking, distances and directions are indispensable properties of geographic information.

4.3.1 Objectives of the Experiment

A case study was conducted using the "search and find nearby places" feature (see Figure 4.10) of Google Maps. The number of location-based online searches has increased rapidly over the past few years. One in five searches is location-related [47] now. Google declared that location-related search helps businesses to make their advertising reach the right customers [48].

4.3.2 Derived Metamorphic Relations

$MR_{SeeEachOther}$: In some software system, users can reasonably expect that if they can find B from A then they can also find A from B, where A and B denote two entities in the system.

4.3.3 Experimental Design

The test followed the steps provided by Google (see Figure 4.10). First, the *current location* to a Hilton hotel in a city was entered. It is possible that many hotels belong

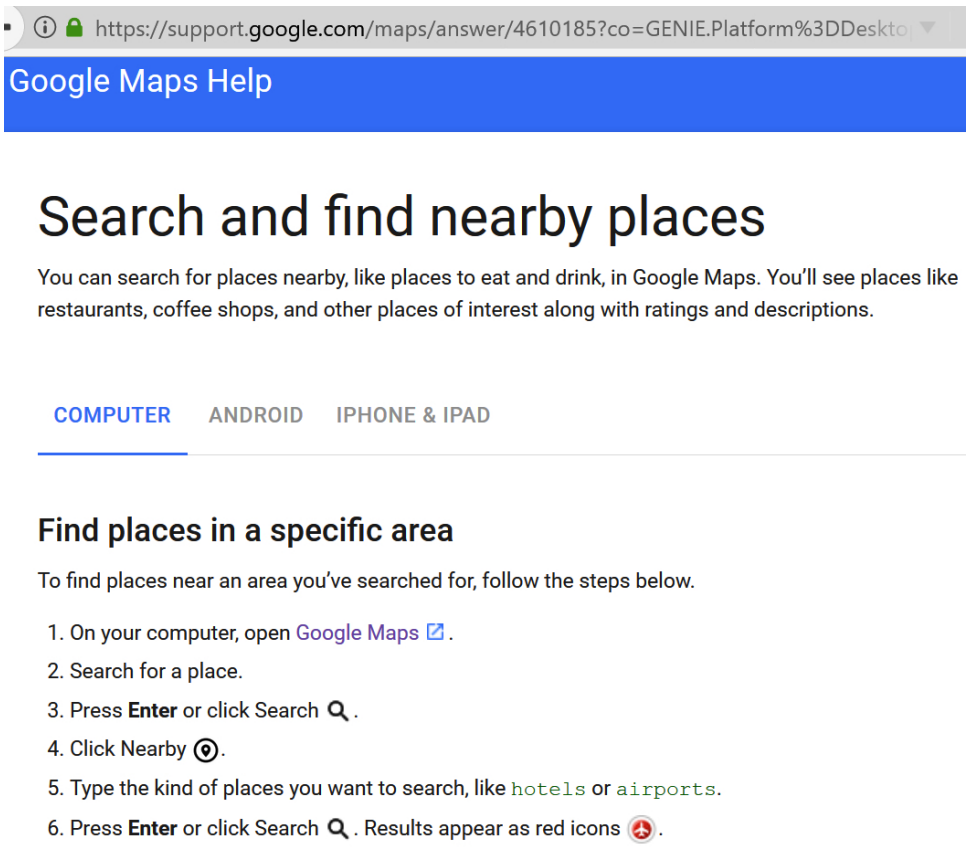


Figure 4.10: Search and find nearby places: A screenshot taken from the Google Maps online help page.

to Hilton, this research randomly choose one. Second, using Hilton as the location, a search for "nearby" cafes (using [cafe] as the query term) was implemented. Let A = $\{a_1, a_2, ..., a_n\}$ denote the set of request results (cafes near the Hilton hotel) [e]. Next, for each $a_i \in A$, the *current location* was set to a_i and then "nearby" cafes was requested again. Let $B_i = \{b_{i,1}, b_{i,2}, ...b_{i,n_i}\}$ denote the set of these request results, that is, the set of cafes nearby. Finally, the following pairs of cafes for each subject city were retrieved:

$$(a_1, b_{1,1}), (a_1, b_{1,2}), ..., (a_1, b_{1,n_1}),$$

$$(a_1, b_{2,1}), (a_1, b_{2,2}), ..., (a_1, b_{2,n_2}),$$

$$...,$$

$$(a_n, b_{n,1}), (a_n, b_{n,2}), ..., (a_n, b_{n,n_n}),$$

In this experiment, 100 pairs of cafes for Hong Kong and another 100 pairs for London were used. For each pair of cafes (a, b), the *current location* was set to b and then a search

[e]Google Maps could return "nearby" places that are located far away in different countries. In all experiments of location-based search, we inspected the search results to exclude such extreme cases from consideration.

for "nearby" cafes was initiated to see whether a was included in the results list. If the result was positive, then a and b could find each other; if the result was negative, then a could find b, but b could not find a, that is, $MR_{SeeEachOther}$ is violated.

4.3.4 Experimental Results

It was found that 27% of the pairs violated the MR in London and the rate for Hong Kong was 5%. These results indicate that the Google Map's location-based search service is asymmetric. For instance, Google Maps can find a "nearby" cafe named Rainforest Cafe (20 Shaftesbury Ave, London) based on the location of St. James Cafe (41 Pall Mall, London) but could not find the St. James Cafe from Rainforest Cafe.

4.3.5 Recommended User Countermeasures

A hypothesis that may explain the asymmetry of Google location-based search was developed. It provides some workarounds to find the blind spot.

The distributed system Google infrastructure is using consists tens of thousands of independent servers. The data in these servers may be different because of an asynchronous problem. To investigate the possibility of exploiting the lack of homogeneity in the Google Maps system, a series of experiments was conducted by leveraging Virtual Private Network (VPN) technology. Two users who are interesting in finding nearby HSBC in 58 Australian towns were simulated. One user visited ".com" domain at maps.google.com at the network in the US and another user issued a query to ".com.au" domain with a IP in Australia. The test results show that only 25 of the 58 towns had the same numbers of results. On one hand, for 15 towns, the user "in the territory of the US" obtained more results. On the other hand, the user located in Australia retrieved more results. In this case, the combination of two search profiles would arguably provide results that are more complete. Some users cannot access VPN, but are still able to issue the same query in different platforms (for example, mobile app and desktop browser) to maximise the results list.

4.4 Case Study of Image Analysis Using MATLAB, OpenCV, and Facebook

With artificial intelligence, face recognition certainly becomes an important research topic and an exciting challenge for modern society. The ability to recognise faces plays a key role in every aspect of life. The process involves complex steps, such as detection, verification and mapping. Nowadays, how to improve the accuracy of facial recognition is an

important research area. This section shows how to use the MRIPs of changing direction to evaluate the recognition of two-dimensional face images.

4.4.1 Objectives of the Experiment

Facebook

Facebook [49] is a popular social networking service. It allows a user to share a post with a photo (see Figure 4.11(a)) and automatically scan the face in it. When user tags photo, if one face is detected, it gives a hint "Who is this?" and draws a rectangle on the position of face (see Figure 4.11(b)).

Matlab

Matlab is a well-known application for science and engineering with abundant toolboxes. In this experiment, the cascade object detector [50] in the Matlab(version R2017b 64-bit) computer vision system toolbox was used to detect faces in photo. Photos are read by the function "imread" and the function "flip" is used to obtain a flipped photo.

OpenCV

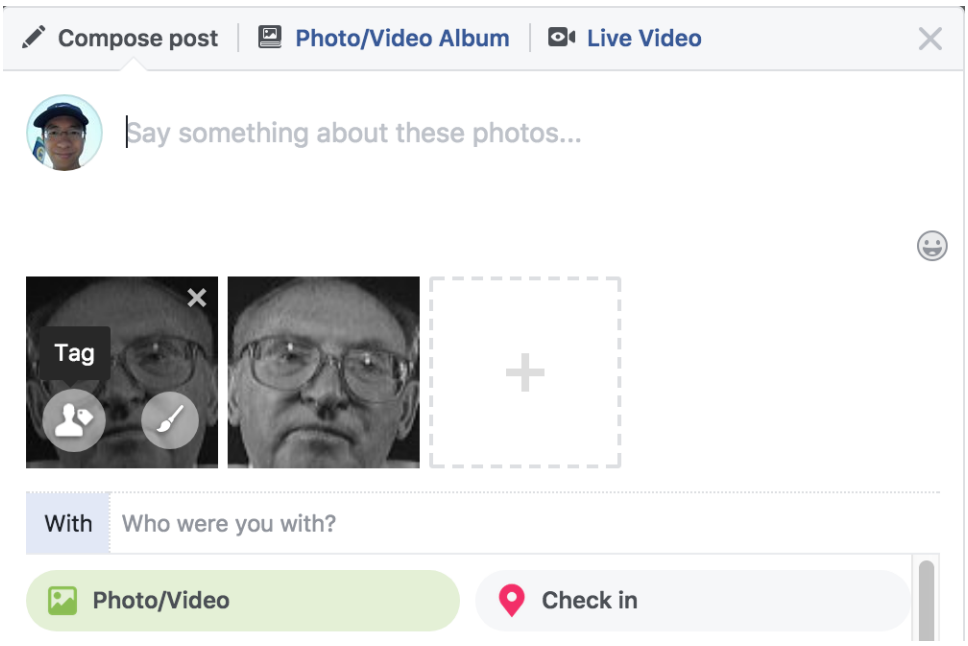
OpenCV (Open Source Computer Vision Library) is a famous open source computer vision and machine learning software library. OpenCV (version 3.3.0) was used with python interface for this empirical study. The function used was cv2.CascadeClassifier.detectMultiScale [51] with pre-trained classifiers configure file [52] for frontal face. All parameters are default values. Photos are read by function "cv2.imread" and are flipped by function "cv2.flip".

4.4.2 Derived Metamorphic Relations

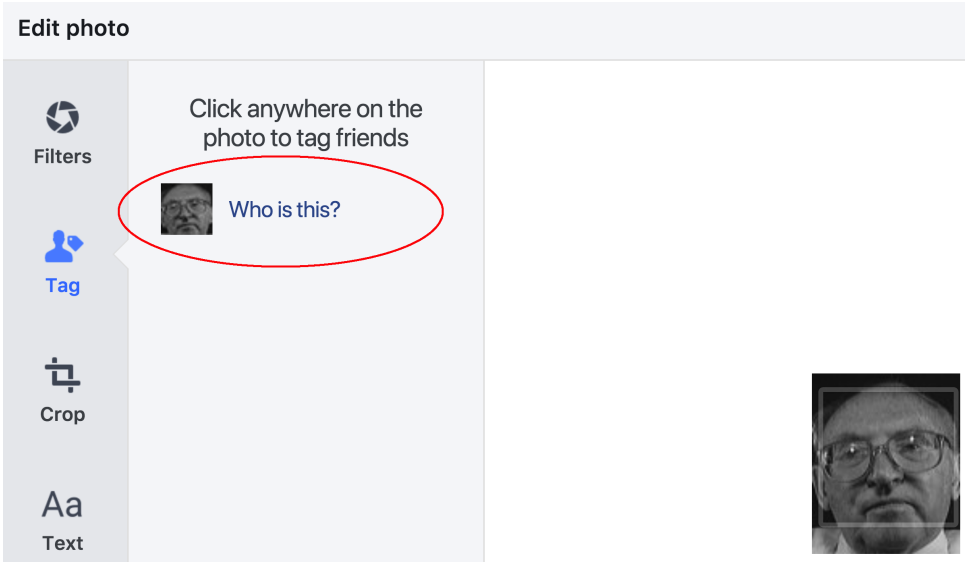
MR_{Flip}: If a mirror image is generated by flipping the original image along the direction of the x-axis, the face in the mirror image is the same face as in the original image, because human faces have approximate bilateral symmetry.

4.4.3 Experimental Design

The database of faces from AT&T Laboratories Cambridge [53] was used. There are 400 test pairs, including ten different photos of each of the 40 people who are in upright, frontal position.



(a) Share post with photo. Click the "Tag" button to edit photo.



(b) Example of face being detected in the uploaded image. It asks user "Who is this?" on the left and draws a semi-opaque box on the face.

Figure 4.11: Face detection on facebook.com

4.4.4 Experimental Results

Table 4.5 summarises the test results. Type 1 failure is that no face was detected in the original image and one face was detected in the flipped image. This category of violation has the highest percentages and all software under test failed this test. Type 2 failure is one face was detected in the original image but none was detected in the flipped one. Only Facebook passed this test. Type 3 violation is two faces were detected in the original image and one face was detected in the flipped image. Both Facebook and OpenCV failed.

Subjects	Test Pairs	Failed Pairs		
		Type 1	Type 2	Type 3
Facebook	400	1	0	1
Matlab	400	24	20	0
OpenCV	400	14	8	1

Table 4.5: Results of face detection

4.5 Case Study of Video Analysis Using Google Cloud Video Intelligence

4.5.1 Derived Metamorphic Relations

$MR_{ReverseTimeline}$: The software should detect and label the same set of objects in the video regardless of playing forwards or backwards.

4.5.2 Objectives of the Experiment

Google Cloud Video Intelligence is a machine learning API [54] which can automatically recognise objects in videos. This experiment involved our experiment on a demo page [55] provided by Google for testing (this demo interface was accessible on Oct 2, 2017, but is unavailable now). It can show detected objects and the related confidence from a video uploaded by the user.

4.5.3 Experimental Design

A 39 seconds video (see Figure 4.13) was shot using mobile phone and ffmpeg (version 3.1.1) [56] was used to create a reversed one. There are five objects in the video: a cup, a keyboard, a knife, a hair dryer and a headphone.

4.5.4 Experimental Results

There are four detected objects from the original video (see Figure 4.14(a)) and five detected objects from the reversed video (see Figure 4.14(b)). Left labels are the name of recognised objects and right numbers reflects the confidence that the label is accurate. Its range is [0, 1].

(a) Facebook Type 1: original photo

(b) Facebook Type 1: flipped photo

(c) Facebook Type 3: original photo

(d) Facebook Type 3: flipped photo

(e) Matlab Type 1: original photo

(f) Matlab Type 1: flipped photo

(g) Matlab Type 2: original photo

(h) Matlab Type 2: flipped photo

(i) OpenCV Type 1: original photo

(j) OpenCV Type 1: flipped photo

(k) OpenCV Type 2: original photo

(l) OpenCV Type 2: flipped photo

(m) OpenCV Type 3: original photo

(n) OpenCV Type 3: flipped photo

Figure 4.12: Example of failure test pairs

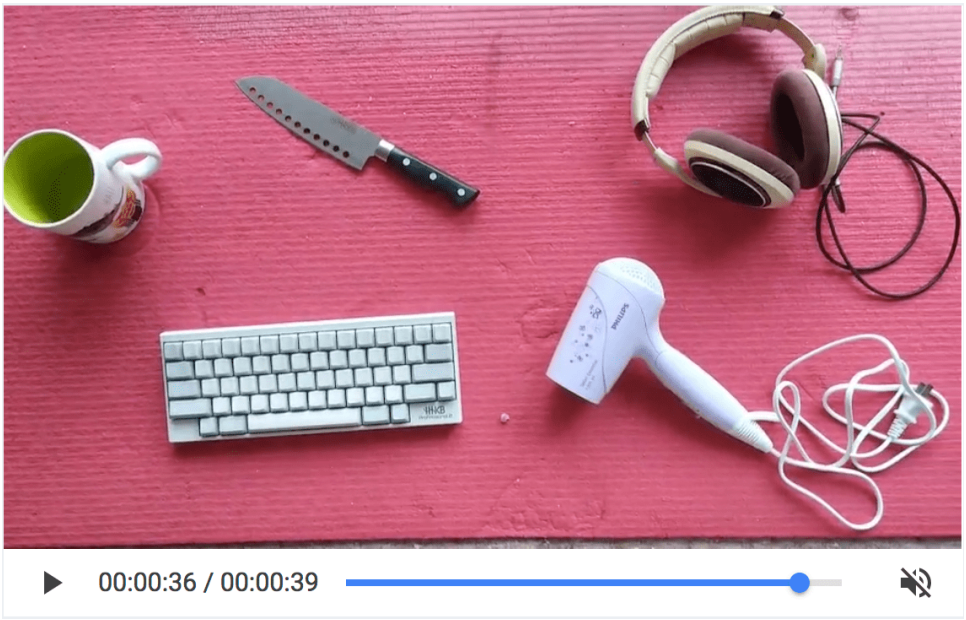
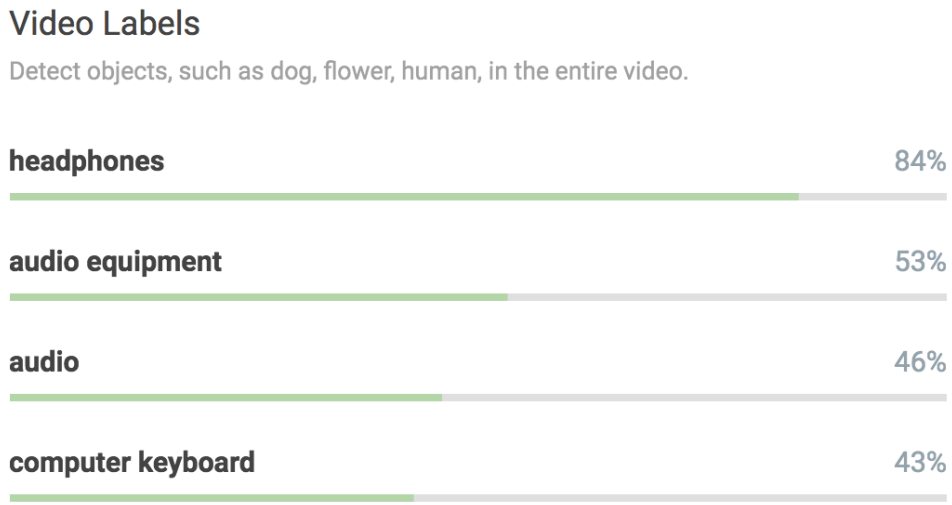
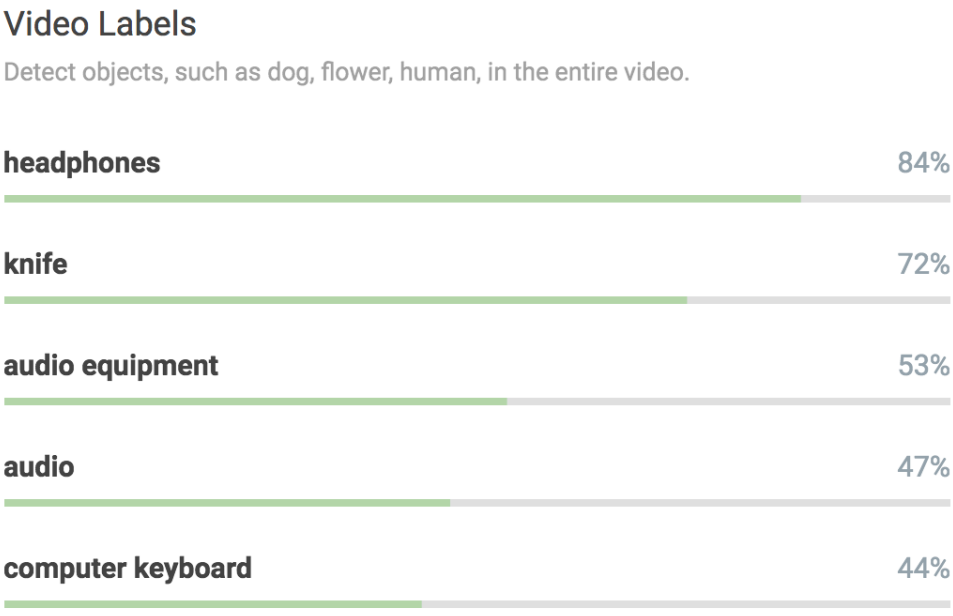


Figure 4.13: Screenshot of test video



(a) Detected objects of original video



(b) Detected objects of reversed video

Figure 4.14: Results of Google Cloud Video Intelligence

Chapter 5

Empirical Evaluation of the Replacement MRIP

5.1 Background

Machine translation is a wide-spread service to replace humans by translating text or speech from one language to another language automatically. At present, many free online translation services are provided by large information technology companies who are competing for users. Each improvement of machine translation services must be evaluated to measure the performance and progress in development. Hence, the better evaluation metrics facilitates machine translation services.

Generally speaking, determination of machine translation quality depends on human intervention. For example, human-quality reference translations are indispensable when it comes to translation quality indicators, and the quantities of precise oracle translations are very limited compared to the possibility of the combination of words.

The absence of test oracle, caused by the flexibility and subjective nature of human language, makes automatic evaluation of machine translation difficult. Zheng et al. [57] presents a new oracle-free approach to detect under-translation and over-translation issues. In particular, two algorithms the author used to address the issues are derived from recommendation algorithm of Item-based Collaborative Filtering and word frequency, both leveraging the large scale of parallel bilingual corpora.

5.2 Objectives of the Experiment

To show the effectiveness of the proposed method, two popular translation services, namely, Google Translate (`https://translate.google.com.au`) and Microsoft Translator (`https://www.bing.com/translator`) were evaluated. Our tool calls their APIs to achieve the English to Chinese translation. According to Google blog, as of 2016, 500

million users were using Google Translate and Microsoft Translator supports more than 60 languages.

5.3 Derived Metamorphic Relation

The identification of a valid and efficient metamorphic relation is a pivotal and challenging work in metamorphic testing. The metamorphic relation presented in this study is motivated by a real world inherent property, namely replacement. More specifically, if an unrelated factor is replaced then the outcome should not be changed. For software testing, replacing uncorrelated input should not influence the output. For example, vehicle colour is unlikely be considered as a property by the object classification module in a self-driving car. It is reasonable to replace an object's colour and expect the same classification result. A fault in the module is revealed if it can recognise a red sedan correctly but mistake a green sedan for a tree.

Following the prospering of networking service, the demand for evaluation at sentence level is rising. However, automatic evaluation measures lack the reliability of a human assessor on individual sentences.

$MR_{ReplaceNoun}$: To simplify the evaluation of sentence structure similarity, a short sentence with subject-verb-object structure was constructed and the subject and object were replaced with a variety of nouns. If the subject is replaced, the translated object is expected to be the same, and vice versa.

Figure 5.1 shows an instance of a defect found in Google Translate by this metamorphic relation. Changing subject from Mike to Mouse leads to a translation inconsistency of a famous restaurant chain brand. Even without the oracle which is the official brand name in Chinese, the approach still can detects the issue.

5.3.1 Experimental Design

A tool in Python was developed to conduct the experiment. All tests were finished in March, 2018.

To generate the subject-verb-object sentence and to ensure the subject nouns are unrelated to the object noun, the top 100 US popular female names in 2016 were selected from the U.S. Social Security Administration website [58]. The 100 top brands around the world in 2015 [59] were chosen as object noun. "Likes" and its antonym "Hates" were chosen as the verb.

For Google Translate, there are $100 \times 100 = 10,000$ English-to-Chinese translations generated for the verb "likes," and same number of translations were executed for the verb "hates". Using Microsoft Translator, the same 20,000 English sentences were translated into Chinese.

(a) "KFC" is translated into "肯德基" in Chinese

(b) "KFC" is translated into "肯德鸡" in Chinese

Figure 5.1: Violation of $MR_{ReplaceNoun}$ in Google Translate. (Note that the word "love" instead of "loves" is used to test the robustness of the translator in dealing with minor grammatical errors.)

5.3.2 Experimental Results

Table 6.1 summarises the test results.

translator	Google Translate		Microsoft Translator	
Verb	likes	hates	likes	hates
number of unique comparisons	990000	990000	990000	990000
noun inconsistency	137438	159044	34670	36779
inconsistency rate (%)	13.88	16.07	3.50	3.72

Table 5.1: Test results

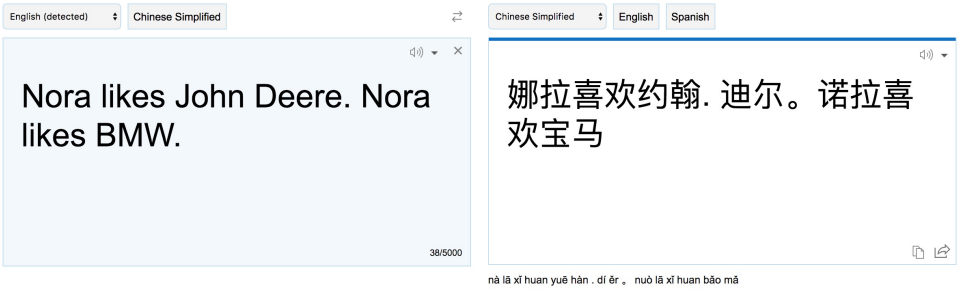
Inconsistent translation issue

When translating English names to Chinese characters, the common method is pronunciation-based transliteration. It is inevitable that multiple Chinese candidate names correspond to a single English name because of the nature of Mandarin. Figure 5.2a demonstrates the inconsistency of translated names in Microsoft Translator. The Chinese name for Nora changes from "娜拉" to "诺拉" when we replace the brand from "John Deere" to "BMW".

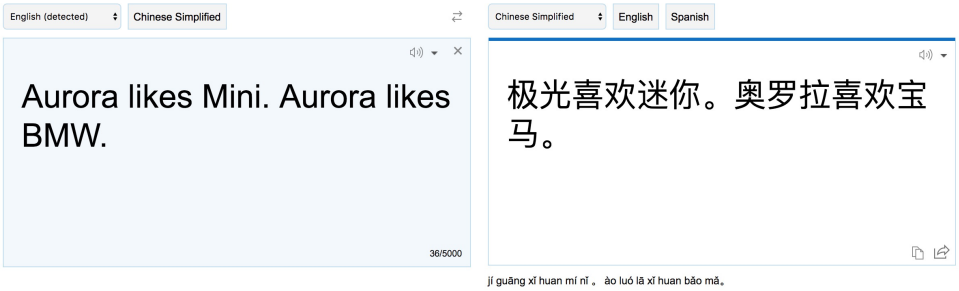
When English names are also words, violation of transliteration is identified by this

approach, shown as Figure 5.2b. “Aurora” is translated to “极光” which is an atmospheric phenomenon when the object is ”BMW”.

These two anomalies are also found when translating brand name. Being different from English names, an appropriately translated brand name is important for business. From a marketing perspective, erroneous literal translation of a brand name not only confuses existing customers, but may also direct potential buyers to an inauthentic product.



(a) Inconsistent translation for English name



(b) English name is not translated by transliteration

Figure 5.2: Inconsistency of translation in Microsoft Translation

Failed-to-translate issue

Figure 5.3 illustrates that the translation services do not translate “Layla” and “Mini” to Chinese for brand and name after replacing. It is possible that a word in the source language is not translated to the target language if it has been recently added to the dictionary. However, another output in the target language rules out this possibility. We are unable to confirm this issue without further knowledge of the system design, but service providers who have more details should be able to decide if it is a failure of translation.

(a) Non translated English name

(b) Not translated brand name

Figure 5.3: Not translated brand and name by Google Translate

Chapter 6

Empirical Evaluation of the Perturbation MRIP

6.1 Objectives of the Experiment

Over the past few years, a large number of autonomous machine projects, such as self-driving cars, drones and advanced robotics, have emerged from across the world showing the prosperity of AI industry. Hence, the safety of self-driving car attracts considerable public attention [60] because the risk and uncertainty brought on by these immature technologies cannot be ignored in the near future. In this study, the software under test is Baidu's Apollo, open source software which provides users a turnkey self-driving car solution. This study's approach was applied to Apollo's perception module (`http://apollo.auto/platform/perception.html`) that includes two parts: 3D Obstacle Perception and Traffic Light Perception. The 3D Obstacle Perception has three sub-systems, namely RADAR Obstacle Perception, LiDAR Obstacle Perception, and Obstacle Results Fusion. Although the experiment was only conducted on the LiDAR Obstacle Perception, the MRIP is also applicable to other perception modules.

The LiDAR Obstacle Perception (LOP) is a subsystem whose input is the 3D point cloud data from the LiDAR hardware (Velodyne's HDL64E LiDAR sensor) and output is a list of detected obstacles. These detected obstacles play a key role because the self-driving car makes decisions and plan according to them. If the information about obstacles around the self-driving car is incomplete or wrong, the decision may cause serious and fatal consequences.

Apollo's perception module (`http://apollo.auto/platform/perception.html`) was tested. The model has two key components: "three-dimensional obstacle perception" and "traffic-light perception". The 3D Obstacle Perception component was tested. It consists of three subsystems. The testing method is applicable to all three subsystems but this study only tested the first subsystem, LiDAR Obstacle Perception (this subsystem

will be referred to as LOP hereafter). The LOP takes the 3D point cloud data from the LiDAR sensor as input, generated by

The LOP processes the raw sensor data using the following pipeline (the following are excerpts from the Apollo website `https://github.com/ApolloAuto/apollo/blob/master/docs/specs/3d_obstacle_perception.md`):

1. HDMap Region of Interest (ROI) Filter (**tested in the experiments**): The Region of Interest (ROI) specifies the drivable area that includes road surfaces and junctions that are retrieved from the HD (high-resolution) map. The HDMap ROI filter processes LiDAR points that are outside the ROI, removing background objects, e.g., buildings and trees around the road. What remains is the point cloud in the ROI for subsequent processing.

2. Convolutional Neural Networks (CNN) Segmentation (**tested in the experiments**): After identifying the surrounding environment using the HDMap ROI filter, Apollo obtains the filtered point cloud that includes only the points inside the ROI (i.e., the drivable road and junction areas). Most of the background obstacles, such as buildings and trees around the road region, have been removed, and the point cloud inside the ROI is fed into the segmentation module. This process detects and segments out foreground obstacles, e.g., cars, trucks, bicycles, and pedestrians. Apollo uses a deep CNN for accurate obstacle detection and segmentation. The output of this process is a set of objects corresponding to obstacles in the ROI.

3. MinBox Builder (**tested in the experiments**): This object builder component establishes a bounding box for the detected obstacles. The main purpose of the bounding box is to estimate the heading of the obstacle (e.g., vehicle) even if the point cloud is sparse. Equally, the bounding box is used to visualise the obstacles.

4. HM Object Tracker (**not tested in the experiments**): This tracker is designed to track obstacles detected by the segmentation step.

5. Sequential Type Fusion (**not tested in our experiments**): To smooth the obstacle type and reduce the type switch over the entire trajectory, Apollo utilises a sequential type fusion algorithm.

Our experiments involved (1), (2) and (3), but not (4) and (5), because the first three parts are the most important and will significantly influence the output of follow-up procedures.

6.2 Derived Metamorphic Relation

The following metamorphic relation was identified. The software under test was the LiDAR Obstacle Perception subsystem (LOP); A and A' denote two inputs to LOP, and O and O1 denote LOP's outputs for A and A', respectively:

MR_{Noise}: Let A and A' be two frames of 3D point cloud data that are identical except that A' includes a small number of additional LiDAR points randomly scattered in regions outside the *HDMap Region of Interest (ROI) Filter*. According to the specification on Apollo's website, ROI specifies the drivable area that includes road surfaces and junctions by removing background objects such as buildings and trees around the road. Each frame of cloud points normally has more than 100,000 points generated by the LiDAR sensor and a small number of additional points (no more than 1000) is trivial.

Let O and O' be the set of obstacles identified by LOP for A and A', respectively (LOP only identifies obstacles within the ROI). Then, the following relation must hold: $O \subseteq O'$

In MR_{Noise}, the added LiDAR points in A' could be small particles in the air, or some noise points from the sensor [61]. These additional LiDAR points are outside the ROI and they intuitively should not affect the detection of obstacles from user perspective. For example, a fly moving between roadside flowers is not supposed to cause a vehicle or bicycle inside the ROI to be missed.

In March 2019, a group of researchers from Tencent Keen Security Lab revealed flaws in the DNN lane recognition function of Tesla Model S by using some small and unobtrusive markings deployed in the real world [62]. Their results show that the noise dots around the lane could make the lane invisible to Tesla's Autopilot. More seriously, three small stickers on the ground could mislead the car into the lane of oncoming traffic. This experiment shows that random spots or noise can indeed put the car at risk in the physical world.

6.2.1 Experimental Design

The dataset "demo-sensor-demo-apollo-1.5.bag" was downloaded from the Apollo Data Open Platform (`http://data.apollo.auto`). This dataset included point raw point cloud data and other sensor data, from real world scenarios. All frames of point cloud data was extracted from the bag file and then 1000 frames were randomly selected. Each frame is a source test case. For each frame, the LOP was performed to obtain its ROI and detected obstacles. Then three follow-up test cases were generated by adding randomly 10, 100 and 1000 points to the three-dimensional space outside the ROI. The minimum and maximum values of additional points for x, y and z coordinate are consistent with the cloud points in the source test case. The next step was to execute the follow-up test case and compare the results of obstacles between the source test case and follow-up test case. Hence, there are a total of 1000 $\times$ 4 = 4000 test cases (1000 source test cases and 3000

follow-up test cases).

6.2.2 Experimental Results

This study only compared the numbers of detected obstacles in O and O', denoted by $|O|$ and $|O'|$, respectively. This is because the track algorithm to check if the object is the same one between two frames outside the scope of our experiment.

Table 6.1 summarises the results. The violation rates (that is, cases for $O > O'$ out of 1000 pairs of outputs) are 2.7% ($= 27 \div 1000$), 12.1% ($= 121 \div 1000$), and 33.5% ($= 335 \div 1000$), for $n = 10$, 100, and 1000, respectively. This means that, as few as 10 sheer random points dispersedly scattered in the vast 3D space outside the ROI can cause the autonomous car to fail to detect an obstacle on the roadway, with a 2.7% probability. When the number of random points increases to 1000, the probability becomes as high as 33.5%.

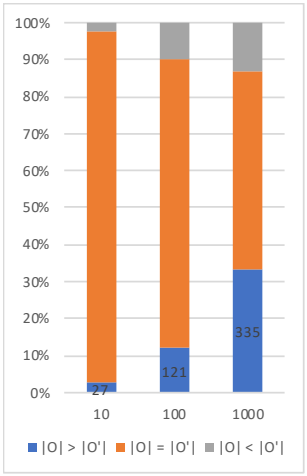
number of added points (n)	|O| > |O'|	|O| = |O'|	|O| < |O'|
10	27	951	22
100	121	781	98
1000	335	533	132

Table 6.1: Summary of test results (for each value of n, 1000 comparisons were made).

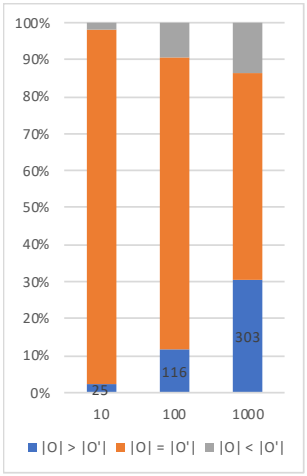
The Apollo defined the detected obstacles into four categories: "detected car, pedestrian, cyclist and unknown are depicted by bounding boxes in green, pink, blue and purple respectively" (`http://apollo.auto/platform/perception.html`). Figures 6.1b to 6.1e summarise the test results of these four categories, and Figure 6.1a shows the overall results corresponding to Table 6.1.

The blue subsections in each vertical bar of Figure 6.1 indicate the number of $|O| > |O'|$ test cases. If the numbers are non-zero, it means critical erroneous implementations are revealed.

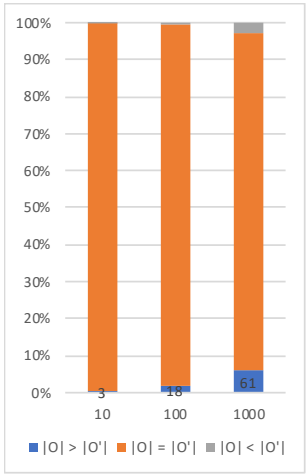
Figures 6.2a and 6.2b show an example where three vehicles could not be detected after adding 1000 random points outside the ROI. Figures 6.2c and 6.2d demonstrate another example, where a pedestrian (the Apollo system depicted this pedestrian using the small pink mark as shown in Figure 6.2c) could not be detected after adding only 10 random points (as shown in Figure 6.2d, the small pink mark is missing). We reported this bug to the Baidu Apollo self-driving car team (`https://github.com/ApolloAuto/apollo/issues/3341`). The Apollo team confirmed the issue by acknowledging, "it might happen" and suggested that "for cases like that, models can be fine-tuned using data augmentation". In other words, the test cases derived by MRIP could also be used as a training set for machine learning.



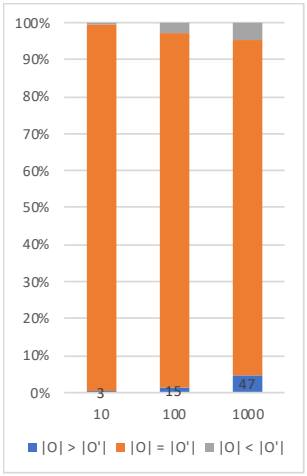
(a) total



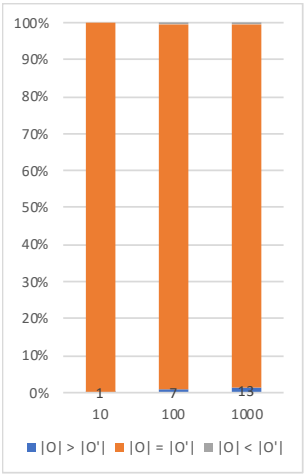
(b) car



(c) pedestrian



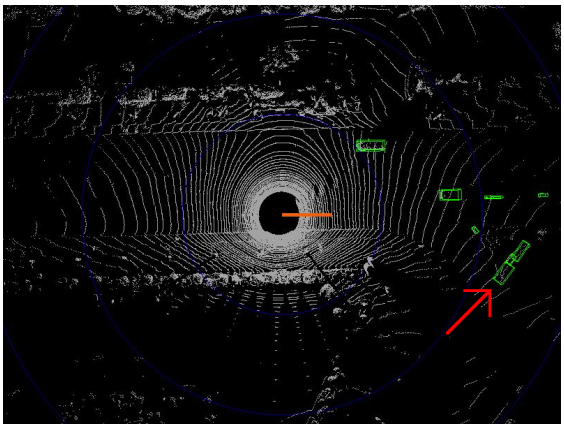
(d) cyclist



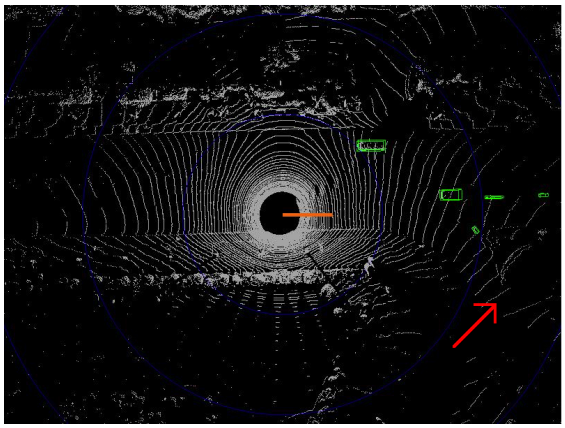
(e) unknown

Figure 6.1: Results of experiments by category: 100% stacked column charts. Each vertical column represents 1000 comparison results, where the blue subsection represents MR_{Noise} violations. Each blue subsection is labelled with the actual number of $|O| > |O'|$ cases (out of the 1000 comparisons).

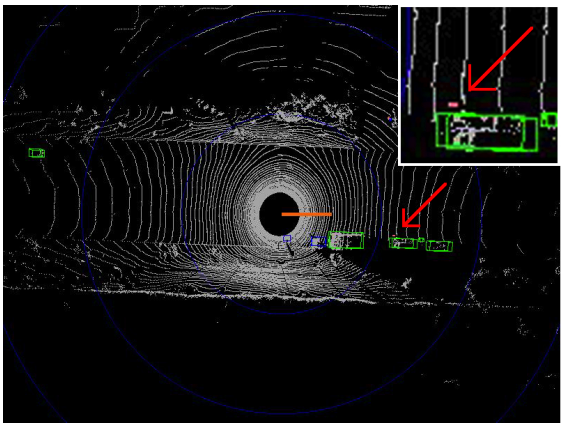
Furthermore, we observed cases where the *type* of the detected obstacle changed after some random noise points were introduced. According the Uber accident preliminary investigation report (see "Preliminary Report Highway: HWY18MH010" at `https://www.ntsb.gov/investigations/AccidentReports/Pages/HWY18MH010-prelim.aspx`), the pedestrian was classified as an "unknown object", then changed to a "vehicle" and then changed to a "bicycle" when the Uber car was approaching. As a result of this uncertainty and change of the obstacle type over a short period of time, the Uber vehicle did not do anything prior to the collision. We detected the "type change" problem in the Baidu Apollo system many days before this Uber accident.



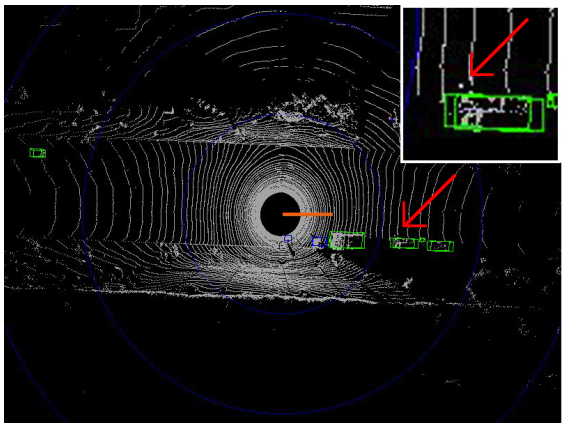
(a) Original (101,676 LiDAR data points; green boxes were generated by the Apollo system to depict the detected cars).



(b) After adding 1000 random data points outside the ROI, the three cars could no longer be detected.



(c) Original (104,251 LiDAR data points; the small pink mark was generated by the Apollo system to depict a detected pedestrian).



(d) After adding only 10 random data points outside the ROI, the pedestrian could no longer be detected.

Figure 6.2: MT detected real-life fatal errors in LiDAR point cloud data interpretation within the Apollo *perception* module: three missing cars and one missing pedestrian.

Chapter 7

Discussions and Conclusion

7.1 Usefulness of MRIPs

MRIP has a focus on input relations. Previous studies on abstract forms of MRs [31, 32, 33] were not focused on input. Nevertheless, in the context of automated test case generation, there is a related technique called data mutation [63]: Given a set of test cases (seeds), data mutation generates new test cases by modifying the seeds using a set of operators, called data mutation operators. Data mutation operators can include, for example: increase or decrease the value of a parameter; set a parameter to 0, to a very large value, or to its negative; swap the values of some parameters; etc. In the extreme case, the modification to the values can be done randomly, which is fuzzing (fuzz testing). Data mutation is different from metamorphic testing as it only considers ways of changing the values of input parameters without looking at the output relations. Technically speaking, metamorphic relations involve semantics rather than data mutation at the syntax level. Nevertheless, data mutation and metamorphic testing can be combined, and have produced practical tools [64]. Data mutation operators can provide heuristics for the identification of potential MRIPs, which will be a focus for our future research.

The vast majority of users need some guidance for identifying suitable MRs. Our concept of patterns aims to help such users to narrow down the search space. In particular, the "change direction" MRIP can help the users and testers to identify different viewpoints towards the system under test. Compared with data mutation operators, the MRIP concept is at a higher level of abstraction. The pattern concept is simple and general enough that it can be easily applied by users to a variety of application domains. The identification of MRIPs can be achieved through levels of abstraction, where a pattern at a high level of abstraction can be first identified and used to derive other patterns at lower levels in the hierarchy, or concrete input relations or MRs at the bottom level.

MRIP is an input-driven approach for MR identification (that is, identification of MRs guided by considering how to modify the input). In contrast to this, the process could also

be output-driven, in which the identification of MRs is guided by first considering the output relations—for example, by referring to some metamorphic relation output patterns such as those designed by Segura et al. [33]. Once an output relation is identified, the kinds of changes to be made to the input in order to achieve that output relation can be considered.

It should be noted that different systems exhibit both commonalities and uniqueness. Using the pattern concept to identify MRs has a focus on commonalities, and therefore may have a disadvantage of overlooking the uniqueness of the specific system under test. Testers, therefore, should not solely rely on patterns to identify MRs, but instead they should also consider the specific features of the system. Nevertheless, the use of patterns can at least provide some hints or starting point for practitioners to apply metamorphic testing.

7.2 Conclusion and Future Work

The oracle problem is one of the fundamental challenges in software testing. Metamorphic testing alleviates the oracle problem in a practical way, but research of generation of concrete and effective metamorphic relations in a wide range of application fields is still at an early stage. In most cases, metamorphic relations are identified in an improvised manner and this needs a wealth of experience in a specific domain. This study proposes three categories of abstract input patterns for systematic generation of metamorphic relations without application domain limitations. In the meantime, we derived concrete metamorphic relations from these input patterns and then conducted an empirical study to illustrate the usefulness of these metamorphic relations in real-life applications.

This study makes three main contributions:

1. The study generically elevates metamorphic testing to the level of commonality of the system under test. This differs from most former studies. The input patterns developed in this research complement the output patterns proposed by Segura et al. [33], and we have extended the pattern concept beyond single application domains. Concrete metamorphic relations are derived from these MRIPs.

2. This is a large-scale empirical study across a number of areas and disciplines. The results show that concrete metamorphic relations can not only reveal previously unknown failures, but also provide objective evaluation to better understand a system.

3. Besides accommodating requirements for testers and developers, countermeasures that take advantage of metamorphic relations are proposed for end users who do not have any domain experience or related skills.

In this thesis, we only investigated one application for each of the following MRIPs: replacement and perturbation. Future research will continue to apply this approach to other application fields and focus on more MRIPs than the three input patterns in this study. Apart from this, an investigation into the relationship between software testing and the properties of systems is anticipated. Moreover, it is anticipated that there will also be a study of the automation of the identification of metamorphic relations based on the pattern concepts.

Bibliography

[1] B. Hailpern and P. Santhanam, "Software debugging, testing, and verification," *IBM Systems Journal*, vol. 41, no. 1, pp. 4–12, 2002.

[2] E. J. Weyuker, "On testing non-testable programs," *The Computer Journal*, vol. 25, no. 4, pp. 465–470, 1982.

[3] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015.

[4] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: A new approach for generating next test cases," Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, Tech. Rep. HKUST-CS98-01, 1998.

[5] T. Y. Chen, T. H. Tse, and Z. Q. Zhou, "Fault-based testing without the need of oracles," *Information and Software Technology*, vol. 45, no. 1, pp. 1–9, 2003.

[6] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. H. Tse, and Z. Q. Zhou, "Metamorphic testing: A review of challenges and opportunities," *ACM Computing Surveys*, vol. 51, no. 1, pp. 4:1–4:27, 2018.

[7] S. Segura, D. Towey, Z. Q. Zhou, and T. Y. Chen, "Metamorphic testing: Testing the untestable," *IEEE Software*, DOI: 10.1109/MS.2018.2875968.

[8] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 805–824, 2016.

[9] T. Y. Chen, D. H. Huang, T. H. Tse, and Z. Q. Zhou, "Case studies on the selection of useful relations in metamorphic testing," in *Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JISIC'04)*. Polytechnic University of Madrid, 2004, pp. 569–583.

[10] T. Y. Chen, F.-C. Kuo, D. Towey, and Z. Q. Zhou, "A revisit of three studies related to random testing," *Science China Information Sciences*, vol. 58, pp. 052 104:1–052 104:9, 2015.

[11] J. Regehr, "Finding compiler bugs by removing dead code," http://blog.regehr.org/archives/1161, June 20, 2014.

[12] A. F. Donaldson, H. Evrard, A. Lascu, and P. Thomson, "Automated testing of graphics shader compilers," *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, pp. 93:1–93:29, 2017.

[13] T. Y. Chen, F.-C. Kuo, W. Ma, W. Susilo, D. Towey, J. Voas, and Z. Q. Zhou, "Metamorphic testing for cybersecurity," *Computer*, vol. 49, no. 6, pp. 48–55, 2016.

[14] J. Brown, Z. Q. Zhou, and Y.-W. Chow, "Metamorphic testing of navigation software: A pilot study with Google Maps," in *Proceedings of the 51st Annual Hawaii International Conference on System Sciences (HICSS-51)*, 2018, pp. 5687–5696, available: http://hdl.handle.net/10125/50602.

[15] X. Xie, J. W. K. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Testing and validating machine learning classifiers by metamorphic testing," *Journal of Systems and Software*, vol. 84, pp. 544–558, 2011.

[16] Z. Q. Zhou, S. Xiang, and T. Y. Chen, "Metamorphic testing for software quality assessment: A study of search engines," *IEEE Transactions on Software Engineering*, vol. 42, no. 3, pp. 264–284, 2016.

[17] M. Jiang, T. Y. Chen, F.-C. Kuo, D. Towey, and Z. Ding, "A metamorphic testing approach for supporting program repair without the need for a test oracle," *Journal of Systems and Software*, vol. 126, pp. 127–140, 2017.

[18] T. Y. Chen, C.-a. Sun, G. Wang, B. Mu, H. Liu, and Z. Wang, "A metamorphic relation-based approach to testing web services without oracles," *International Journal of Web Services Research*, vol. 9, no. 1, pp. 51–73, 2012.

[19] W. K. Chan, S. C. Cheung, and K. R. P. H. Leung, "A metamorphic testing approach for online testing of service-oriented software applications," *International Journal of Web Services Research*, vol. 4, no. 2, pp. 60–80, 2007.

[20] D. Towey, H. Liu, T. Y. Chen, F.-C. Kuo, and Z. Q. Zhou, "Metamorphic testing: A new student engagement approach for a new software testing paradigm," in *Proceedings of the IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE '16)*. IEEE, 2016, pp. 218–225.

[21] H. Liu, F. Kuo, and T. Y. Chen, "Teaching an end-user testing methodology," in *2010 23rd IEEE Conference on Software Engineering Education and Training*, 2010, pp. 81–88.

[22] H. Liu, F.-C. Kuo, D. Towey, and T. Y. Chen, "How effectively does metamorphic testing alleviate the oracle problem?" *IEEE Transactions on Software Engineering*, vol. 40, no. 1, pp. 4–22, 2014.

[23] Y. Cao, Z. Q. Zhou, and T. Y. Chen, "On the correlation between the effectiveness of metamorphic relations and dissimilarities of test case executions," in *Proceedings of the 13th International Conference on Quality Software (QSIC '13)*. IEEE, 2013, pp. 153–162.

[24] Z. Q. Zhou, "Using coverage information to guide test case selection in adaptive random testing," in *Proceedings of the 34th Annual International Computer Software and Applications Conference (COMPSAC'10), 7th International Workshop on Software Cybernetics*. IEEE Computer Society Press, 2010, pp. 208–213.

[25] Z. Q. Zhou, A. Sinaga, and W. Susilo, "On the fault-detection capabilities of adaptive random test case prioritization: Case studies with large test suites," in *Proceedings of the 45th Annual Hawaii International Conference on System Sciences (HICSS'12)*. IEEE Computer Society Press, 2012, pp. 5584–5593.

[26] U. Kanewala and J. M. Bieman, "Using machine learning techniques to detect metamorphic relations for programs without test oracles," in *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, 2013, pp. 1–10.

[27] U. Kanewala, J. M. Bieman, and A. Ben-Hur, "Predicting metamorphic relations for testing scientific software: a machine learning approach using graph kernels," *Software Testing, Verification and Reliability*, vol. 26, pp. 245–269, 2016.

[28] T. Y. Chen, P. L. Poon, and X. Xie, "METRIC: METamorphic Relation Identification based on the Category-choice framework," *Journal of Systems and Software*, vol. 116, pp. 177–190, June 2016.

[29] A. Gotlieb and B. Botella, "Automated metamorphic testing," in *Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC'03)*. IEEE Computer Society Press, Los Alamitos, California, 2003, pp. 34–40.

[30] J. Ding and D. Zhang, "An approach for iteratively generating adequate tests in metamorphic testing: A case study," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, 2016, pp. 263–268.

[31] Z. Q. Zhou, T. H. Tse, F.-C. Kuo, and T. Y. Chen, "Automated functional testing of web search engines in the absence of an oracle," Department of Computer Science, The University of Hong Kong, Tech. Rep. TR-2007-06, 2007.

[32] Z. Q. Zhou, S. Zhang, M. Hagenbuchner, T. H. Tse, F.-C. Kuo, and T. Y. Chen, "Automated functional testing of online search services," *Software Testing, Verification and Reliability*, vol. 22, no. 4, pp. 221–243, 2012.

[33] S. Segura, J. A. Parejo, J. Troya, and A. Ruiz-Cortés, "Metamorphic testing of RESTful web APIs," *IEEE Transactions on Software Engineering*, vol. 44, no. 11, November 2018.

[34] F.-H. Su, J. Bell, C. Murphy, and G. Kaiser, "Dynamic inference of likely metamorphic properties to support differential testing," in *Proceedings of the IEEE/ACM 10th International Workshop on Automation of Software Test*. IEEE, 2015, pp. 55–59.

[35] J. Troya, S. Segura, and A. Ruiz-Cortés, "Automated inference of likely metamorphic relations for model transformations," *Journal of Systems and Software, Special Issue on Test Oracles*, vol. 136, pp. 188–208, 2018.

[36] S. Segura, "Metamorphic testing: Challenges ahead (keynote speech)," in *Proceedings of the IEEE/ACM 3rd International Workshop on Metamorphic Testing (MET '18), in conjunction with the 40th International Conference on Software Engineering (ICSE '18)*. ACM, May 2018, pp. 1–1.

[37] J. Larson, S. Mattu, L. Kirchner, and J. Angwin. (2016) How we analyzed the compas recidivism algorithm. [Online]. Available: https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm

[38] L. F. J. D. Almeida and S. R. Vergilio, "Exploring perturbation based testing for web services," in *2006 IEEE International Conference on Web Services (ICWS'06)*, 2006, pp. 717–726.

[39] The Nielsen Company. (2016) IAB and Nielsen launch world-leading total digital audience measurement solution, digital ratings (monthly). [Online]. Available: http://www.nielsen.com/au/en/press-room/2016/iab-and-nielsen-launch-digital-ratings-monthly.html

[40] Alexa Internet, Inc. (2017) Top sites in Australia. [Online]. Available: http://www.alexa.com/topsites/countries/AU

[41] Deloitte. (2017) Global powers of retailing 2017: The art and science of customers. [Online]. Available: https://www2.deloitte.com/global/en/pages/consumer-business/articles/global-powers-of-retailing.html

[42] eBizMBA. (2017, Mar.) Top 15 most popular real estate websites. [Online]. Available: http://www.ebizmba.com/articles/real-estate-websites

[43] ——. (2017, Mar.) Top 15 most popular car websites. [Online]. Available: http://www.ebizmba.com/articles/car-websites

[44] Alexa Internet, Inc. (2017, Mar.) Alexa - top sites by category: Shopping/health/pharmacy/online pharmacies. [Online]. Available: http://www.alexa.com/topsites/category/Shopping/Health/Pharmacy/Online_Pharmacies

[45] ISO/IEC 25010:2011, "Systems and software engineering – systems and software quality requirements and evaluation (SQuaRE) – system and software quality models," 2011.

[46] S.-W. Lee, B. Moon, C. Park, J.-M. Kim, and S.-W. Kim, "A case for flash memory ssd in enterprise database applications," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '08. ACM, 2008, pp. 1075–1086.

[47] A. Gesenhues, "Google searches now correspond to user location instead of domain," *Search Engine Land*, Oct. 27, 2017. [Online]. Available: https://searchengineland.com/google-searches-now-correspond-location-versus-country-services-attached-domain-285769

[48] Google, "Target ads to geographic locations," 2017. [Online]. Available: https://support.google.com/adwords/answer/1722043?hl=en

[49] Facebook. (2017) Facebook. [Online]. Available: https://www.facebook.com

[50] MathWorks. (2017) Detect objects using the viola-jones algorithm - matlab - mathworks australia. [Online]. Available: https://au.mathworks.com/help/vision/ref/vision.cascadeobjectdetector-system-object.html

[51] OpenCV. (2017) Opencv: cv::cascadeclassifier class reference. [Online]. Available: https://docs.opencv.org/3.3.0/d1/de5/classcv_1_1CascadeClassifier.html

[52] ——. (2017) opencv/haarcascade_frontalface_default.xml at master · opencv/opencv. [Online]. Available: https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml

[53] A. L. Cambridge. (2017) The database of faces. [Online]. Available: http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

[54] F.-F. Li. (2017) Announcing Google Cloud Video Intelligence API, and more cloud machine learning updates. Google Cloud Big Data and Machine Learning Blog. [Online]. Available: https://cloud.google.com/blog/big-data/2017/03/announcing-google-cloud-video-intelligence-api-and-more-cloud-machine-learning-updates

[55] Google, Inc. (2017) Cloud video intelligence - video content analysis. [Online]. Available: https://cloud.google.com/video-intelligence/

[56] FFmpeg. (2017) Ffmpeg. [Online]. Available: https://www.ffmpeg.org/

[57] W. Zheng, W. Wang, D. Liu, C. Zhang, Q. Zeng, Y. Deng, W. Yang, and T. Xie, "Oracle-free Detection of Translation Issue for Neural Machine Translation," *ArXiv e-prints*, Jul. 2018.

[58] Social Security, USA. (2018) Popular baby names. [Online]. Available: https://www.ssa.gov/cgi-bin/popularnames.cgi

[59] B. Chapman. (2016) The top 100 brands in the world have been revealed. [Online]. Available: https://www.independent.co.uk/news/business/news/apple-most-valuable-brand-iphone-7-google-coca-cola-a7345501.html

[60] S. Levin, "Uber crash shows 'catastrophic failure' of self-driving technology, experts say," https://www.theguardian.com/technology/2018/mar/22/self-driving-car-uber-death-woman-failure-fatal-crash-arizona, March 23, 2018.

[61] White Paper, "Velodyne's hdl-64e: A high definition lidar sensor for 3-d applications," www.velodynelidar.com, 2007.

[62] E. Ackerman, "Three small stickers in intersection can cause tesla autopilot to swerve into wrong lane," *IEEE Spectrum*, April 1, 2019. [Online]. Available: https://spectrum.ieee.org/cars-that-think/transportation/self-driving/three-small-stickers-on-road-can-steer-tesla-autopilot-into-oncoming-lane

[63] L. Shan and H. Zhu, "Generating structurally complex test cases by data mutation: A case study of testing an automated modelling tool," *The Computer Journal*, vol. 52, no. 5, pp. 571–588, 2009.

[64] H. Zhu, "JFuzz: A tool for automated Java unit testing based on data mutation and metamorphic testing methods," in *Proceedings of the 2nd International Conference on Trustworthy Systems and Their Applications*. IEEE, 2015, pp. 8–15.

Appendix A

Nouns used in $MR_{ReplaceNoun}$

A.1 Subject nouns

This section includes the top 100 US female names used as subject nouns in $MR_{ReplaceNoun}$ experiment.

Emma	Zoey	Anna	Alice
Olivia	Penelope	Hazel	Luna
Ava	Lillian	Aaliyah	Bella
Sophia	Addison	Ariana	Quinn
Isabella	Layla	Lucy	Lydia
Mia	Natalie	Caroline	Peyton
Charlotte	Camila	Sarah	Melanie
Abigail	Hannah	Genesis	Kylie
Emily	Brooklyn	Kennedy	Aubree
Harper	Zoe	Sadie	Mackenzie
Amelia	Nora	Gabriella	Kinsley
Evelyn	Leah	Madelyn	Cora
Elizabeth	Savannah	Adeline	Julia
Sofia	Audrey	Maya	Taylor
Madison	Claire	Autumn	Katherine
Avery	Eleanor	Aurora	Madeline
Ella	Skylar	Piper	Gianna
Scarlett	Ellie	Hailey	Eliana
Grace	Samantha	Arianna	Elena
Chloe	Stella	Kaylee	Vivian
Victoria	Paisley	Ruby	Willow
Riley	Violet	Serenity	Reagan
Aria	Mila	Eva	Brianna
Lily	Allison	Naomi	Clara
Aubrey	Alexa	Nevaeh	Faith

A.2 Object nouns

This section includes the top 100 brands used as object nouns in $MR_{ReplaceNoun}$ experiment.

Apple	IKEA	Allianz	MasterCard
Google	Zara	Siemens	DHL
Coca-Cola	Pampers	Gucci	Land Rover
Microsoft	UPS	Goldman Sachs	FedEx
Toyota	Budweiser	Danone	Harley-Davidson
IBM	J.P. Morgan	Nestlé	Prada
Samsung	eBay	Colgate	Caterpillar
Amazon	Ford	Sony	Burberry
Mercedes-Benz	Hermès	3M	Xerox
GE	Hyundai	adidas	Jack Daniel's
BMW	Nescafe	Visa	Sprite
McDonald's	Accenture	Cartier	Heineken
Disney	Audi	Adobe	Mini
Intel	Kellogg's	Starbucks	Dior
Facebook	Volkswagen	Morgan Stanley	PayPal
Cisco	Philips	Thomson Reuters	John Deere
Oracle	Canon	Lego	Shell
Nike	Nissan	Panasonic	Corona
Louis Vuitton	Hewlett Packard Enterprise	Kia	MTV
H&M	L'Oréal	Santander	Johnnie Walker
Honda	AXA	Discovery	Smirnoff
SAP	HSBC	Huawei	Moët & Chandon
Pepsi	HP	Johnson & Johnson	Ralph Lauren
Gillette	Citi	Tiffany & Co.	Lenovo
American Express	Porsche	KFC	Tesla